НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
"___"_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 " Комп'ютерні науки "

на тему «Моделювання оцінки соціально-політичних ризиків з використанням ГІС технологій»

Виконав (-ла): студент (-ка) 4 курсу, групи ТМ-52

Седлак Микита Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент Швайко В.Г _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року
Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший бакалаврський

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ

на дипломну роботу студенту

Седлаку Микиті Володимировичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Моделювання оцінки соціально-політичних ризиків з використанням ГІС технологій»
керівник роботи Швайко Валерій Григорович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”22” 05 2019р. № 1325С

2. Строк подання студентом роботи .06.2019

3. Вихідні дані до роботи програмний застосунок розроблено у середовищі Visual Studio 2017 на платформі .NET з використанням мови C#.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Створити модель бази даних для моделювання оцінки соціально-політичних ризиків. Проаналізувати існуючі рішення. Розробити застосунок, що дасть змогу працювати зі створеною базою даних та відображати отримані дані на мапі України.

5. Перелік ілюстративного матеріалу

«Задача розробки інструментального засобу», «Актуальність теми», «Поняття демографічної безпеки», «Аналіз існуючих рішень», «Створення програмного засобу», «Взаємодія компонентів», «Функціональна схема системи», «Приклади роботи програми», «Висновки».

6. Дата видачі завдання ”14” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	09.10.2019	
2.	Вивчення та аналіз задачі	14.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	02.02.2019-03.03.2019	
4.	Розробка структур окремих підсистем	04.03.2019-14.04.2019	
5.	Програмна реалізація системи	15.04.2019-19.05.2019	
6.	Оформлення пояснювальної записки	20.05.2019-05.06.2019	
7.	Захист програмного продукту	17.05.2019	
8.	Передзахист	31.05.2019	
9.	Захист	17.06.2019-21.06.2019	

Студент _____ Седлак М.В.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Швайко В.Г.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Обсяг дипломної роботи складає 52 сторінки, 17 рисунків, 3 додатки та 18 посилань.

Метою дипломної роботи є розробка програмного продукту, який надасть користувачу швидко та зручно обробити статистичні дані, записати їх до бази даних та відобразити їх у візуальному вигляді на мапі України.

Під час роботи було порівняно та проаналізовано більшість засобів для побудови геоінформаційних систем, зроблено огляд програмного комплексу ArcGis та описані його основні недоліки. У якості рішення цих недоліків було запропоновано поєднання деяких функцій ArcGis з розробленими інструментами та програмними рішеннями, які були створені під час дипломної роботи. Поєднання було здійснено завдяки потужним можливостям програмної платформи .NET Framework та об'єктно-орієнтованої мови програмування C#.

Результатом роботи стало створення програмного інструментального засобу для обробки та відображення статистичних даних.

Ключові слова: демографічна безпека, ArcGis, національна безпека, редактор, народжуваність, смертність, демографія.

ANNOTATION

Thesis consists of 52 pages, includes 17 figures, 3 annexes and 18 references.

The purpose of the thesis is to develop a software product that will provide user with a quick and convenient way to process statistical data, save it to the database and display it visually on the map of Ukraine.

During the work, the majority of tools for building geoinformation systems were compared and analyzed, a review of ArcGis software was made and It's main disadvantages were described. As a solution for these disadvantages it was proposed to use a combination of some ArcGis features with developed tools and software solutions, that were created during working on this Thesis. The combination has been achieved thanks to powerful capabilities of the .NET Framework and object-oriented C # programming language.

The result of the work was creation of a software tool for processing and displaying statistical data.

Key words: demographic security, ArcGis, national security, editor, birth rate, mortality, demography.

АННОТАЦИЯ

Объем дипломной работы составляет 52 страницы, 17 рисунков, 3 приложения и 18 ссылок.

Целью работы является разработка программного продукта, который предоставит пользователю возможность быстро и удобно обработать статистические

данные, записать их в базу данных и отобразить их в визуальном виде на карте Украины.

Во время работы было сравнено и проанализировано большинство средств для построения геоинформационных систем, сделан обзор программного комплекса ArcGis и описаны его основные недостатки. В качестве решения этих недостатков было предложено сочетание некоторых функций ArcGis разработанным инструментами и программными решениями, которые были созданы во время дипломной работы. Сочетание было осуществлено благодаря мощным возможностям программной платформы .NET Framework и объектно-ориентированного языка программирования C #.

Результатом работы стало создание программного инструментального средства для обработки и отображения статистических данных.

Ключевые слова: демографическая безопасность, ArcGis, национальная безопасность, редактор, рождаемость, смертность, демография.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ..... 65

ВСТУП..... 66

1. ЗАДАЧА РОЗРОБКИ ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ ДЛЯ МОДЕЛЮВАННЯ ОЦІНКИ СОЦІАЛЬНО-ПОЛІТИЧНИХ РИЗИКІВ З ВИКОРИСТАННЯМ ГІС ТЕХНОЛОГІЙ..... 68

1.1 Особливості розробки клієнтської частини програмного засобу 68
1.2 Користувацький інтерфейс системи 69
1.3 Дослідження демографічної безпеки 69
1.3.1 Основні формули розрахунку статичного приросту населення 70
1.3.2 Розрахунок аналізу смертності 72
1.3.3 Природний приріст населення 73
1.3.4 Розрахунок коефіцієнтів шлюбності та розлучуваності 73
1.3.5 Загальний коефіцієнт шлюбності 74
1.3.6 Абсолютні показники розлучення 74
1.3.7 Загальний коефіцієнт розлучуваності 74
1.4 Потенційні користувачі 75
2. АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ ДЛЯ ОЦІНКИ СОЦІАЛЬНО-ПОЛІТИЧНИХ РИЗИКІВ З ВИКОРИСТАННЯМ ГІС ТЕХНОЛОГІЙ 75
2.1 Опис предметної області 76
2.2 Висновки до розділу 77
3. ЗАСОБИ РОЗРОБКИ 78
3.1 Платформа .Net Framework 78
3.1.1 Загальні відомості про платформу .NET Framework 79
3.1.2 Середовище CLR 81
3.2 Мова програмування C# 83
3.3 Плаский дизайн 86
3.4 Середовище розробки Visual Studio 88
3.5 Технологія для розробки інтерфейсу 89
3.6 Бібліотека Entity framework 90
3.7 Середовище Microsoft SQL Server 92
3.8 Середовище ArcGIS 94
3.9 Висновки до розділу 96
4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ 97
4.1 Архітектура додатку для роботи з БД 97
4.1.1 Концептуальна модель 100
4.1.2 Шаблон MVP 101

4.1.3 Шаблон Репозиторій 102
4.1.4 Шар доступу до даних 103
4.1.5 Шар бізнес-логіки 104
4.1.4 Шар інтерфейсу користувача 105
4.2 Висновки до розділу 105
5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ 106
5.1 Системні вимоги та інсталяція 106
5.2 Сценарії роботи з програмним продуктом 107
ВИСНОВКИ 110
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 111

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

UI — графічний інтерфейс користувача.

DB — база даних.

COM — об’єктна модель компонентів.

DLL — бібліотека динамічної компоновки.

API — програмний інтерфейс додатку.

LINQ — запити інтегровані в мову.

MVP — шаблон проектування.

MVC — шаблон проектування.

XML — розширювана мова розмітки.

Валідація — перевірка коректності.

Фреймворк — зовнішній програмний продукт.

ВСТУП

Демографічна безпека — стан демографічних процесів, яких достатньо для відтворення населення без істотного впливу зовнішніх факторів і забезпечення людськими ресурсами геополітичних інтересів держави.

Мета демографічної безпеки — забезпечення регулювання чисельності населення держави зі збереженням етнопропорціональної структури і генетичного здоров'я людей. Це вказує на те, що у різних держав в залежності від ситуації в області народжуваності можуть бути різні завдання. Так, в країнах, що знаходяться на етапі стрімкого розвитку, де спостерігається демографічний вибух, проводиться політика, яка спрямована на обмеження народжуваності, оскільки стрімке зростання населення істотно збільшує навантаження на економіку і систему соціального забезпечення. В умовах депопуляції, навпаки, держава прагне підвищити народжуваність і тривалість життя людей.

Актуальність теми полягає в тому, що зі стрімким розвитком технологій та появою великого обсягу статистичних даних з’явились складнощі з їх опрацюванням, складанням загальної статистики та її відображенням у вигляді, який має інтуїтивно зрозумілий інтерфейс для користувача, який оперує цими даними. Кожний крок вимагає від користувача постійної концентрації та уваги, дані необхідно вводити вручну у поля, які розташовані на різних частинах документу.

Тому метою є розробка програмного додатку, який має необхідні функції для додавання та відображення даних, та в свою чергу зможе полегшити обробку, автоматично уструктуризувати отриману від користувача інформацію.

Програмний додаток розроблявся з використанням мови програмування C#, платформи .NET Framework та програмного комплексу ArcGis.

1. ЗАДАЧА РОЗРОБКИ ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ ДЛЯ МОДЕЛЮВАННЯ ОЦІНКИ СОЦІАЛЬНО-ПОЛІТИЧНИХ РИЗИКІВ З ВИКОРИСТАННЯМ ГІС ТЕХНОЛОГІЙ

Метою розробки є створення програмного додатку, який вимагає від користувача меншої кількості етапів і дій під час введення та відображення статистичних даних. Збільшити зрозумілість і зручність користування шляхом побудови інтуїтивно зрозумілого користувацького інтерфейсу.

Призначенням даного програмного застосунку є надання зрозумілого інтерфейсу для завантаження статистичних даних та відображення їх на мапі України.

Програмний засіб розробляється для персональних комп'ютерів під керівництвом операційної системи Windows XP / 7 / 10.

Вхідні дані: дані отримуються за файлу фіксованої структури з розширенням *.xlsx.

Вихідні дані: дані відображаються у візуальному вигляді з прив'язкою до кожної області і району України.

Розглянути можливість побудови власного програмного застосунку, як новий спосіб обробки та відображення статистичних даних.

Необхідними можливостями, які повинен забезпечити програмний засіб, є:

— завантаження даних з комп'ютера користувача;

— відкриття та зчитування файлів у форматі *.xlsx;

— можливість перегляду завантажених даних;

— декомпозиція та запис даних до бази даних;

— можливість відобразити дані на мапі України.

1.1 Особливості розробки клієнтської частини програмного засобу

Призначення даного програмного засобу є надання чіткого та зрозумілого інтерфейсу для використання іншими частинами системи з забезпеченням повного функціоналу усіх необхідних можливостей. Додаток містить декілька класів, кожен з яких відповідає за свій сервіс та несе певний функціонал.

Загалом додаток містить наступний функціонал:

— відкриття потрібного файлу для завантаження;
— відображення завантаженої інформації;
— запис отриманих даних до БД;
— видалення непотрібних полів;
— відображення повної інформації про певні регіони на карті України.

1.2 Користувацький інтерфейс системи

Головна ідея стилістики додатку — максимальна простота інтерфейсу. Разом з тим дизайн спроектовано в стилі Flat UI[1], що зараз є провідним стилем багатьох застосунків, в тому числі desktop-сервісів. Він базується на принципах швейцарського стилю, в базі якого лежить типографіка як спосіб виключити зайві деталі інтерфейсу. Простота та мінімалізм є основою візуального оформлення такого проекту та забезпечує легкість сприйняття інформації, а також інтуїтивно зрозумілий інтерфейс для користувачів.

1.3 Дослідження демографічної безпеки

Демографічна безпека[2] — такий стан демографічних процесів, якого достатньо для відтворення населення без істотного впливу зовнішнього фактора і забезпечення людськими ресурсами геополітичних інтересів держави.

До демографічних загроз населення належать:

— депопуляція;

— старіння населення;

— нерегульовані міграційні процеси;

— деградація інституту родини.

Індикаторами соціально-економічної безпеки, що мають відношення до цього аспекту і за своїм спрямуванням є:

— загальний коефіцієнт народжуваності;

— загальний коефіцієнт смертності;

— загальний коефіцієнт одруження;

— загальний коефіцієнт розлучення;

— загальний коефіцієнт дитячої смертності (на 1000 осіб).

В основу розрахунків демографічних коефіцієнтів і показників будуть закладені наступні формули і поняття, в подальшому буде їх модифікація для визначення математичної моделі і принципу збереження даних.

1.3.1 Основні формули розрахунку статичного приросту населення

Оскільки для розрахунку відносних показників використовуються абсолютні показники, безпосередньо перед проведенням розрахунку проводиться аналіз вхідної інформації на наявність «невідомих» даних. Наприклад, при розрахунку показників за віком необхідно врахувати, що вік особи, відносно якої відбулась демографічна подія, може бути невідомим. У цих випадках попередньо здійснюється розподіл осіб невідомого віку по вікових групах. При цьому розподіл осіб невідомого віку проводиться пропорційно чисельності вікових груп осіб, вік яких відомий.

Процедура розподілу осіб невідомого віку здійснюється у два етапи.

Спочатку розраховується коефіцієнт розподілу:

$$k = \frac{K}{K - K_n} \quad (1.1)$$

де: k — коефіцієнт розподілу осіб невідомого віку;

K— загальна кількість осіб, з якими відбулася відповідна демографічна подія.

K_n — кількість осіб, з якими відбулася відповідна демографічна подія і вік яких невідомий.

Наступним кроком є розподіл осіб невідомого віку шляхом множення кількості осіб певного віку на коефіцієнт розподілу осіб невідомого віку:

$$K_x = K'_x \times k \quad (1.2)$$

де: K_x — кількість демографічних подій, які відбулися з населенням у віці *x* років, з урахуванням розподілу демографічних подій для осіб, вік яких невідомий.

K'_x— кількість демографічних подій, які відбулися з населенням у віці *x* років, без урахування демографічних подій для осіб, вік яких невідомий, *осіб*;

k — коефіцієнт розподілу осіб невідомого віку.

Процедура розподілу осіб невідомого віку наведена на прикладі розподілу кількості народжених за віком матері. Для цього кількість народжених у матерів, вік яких невідомий, розподіляється пропорційно на всі представлені групи.

Як правило, демографічні показники розраховуються для року в цілому. Проте за необхідністю їх можна обчислити і за інші періоди часу: як більше одного року, так і менше. При цьому існують особливості розрахунку коефіцієнтів інтенсивності, які обчислюються як відношення абсолютної кількості демографічних подій до середньорічної чисельності населення. Загальний вигляд річного коефіцієнта інтенсивності наведено у формулі (1.3):

$$K_i = \frac{NDE}{\bar{S}} \times 1000 \quad (1.3)$$

де: K_i — річний коефіцієнт інтенсивності, *%*;

NDE — кількість демографічних подій (народжень, смертей, шлюбів, розлучень) за рік;

$\bar{S}$— середньорічна чисельність населення, *осіб*.

Розрахунок коефіцієнтів за період менше року проводиться до річної формули за формулою (1.4)

$$K_i^T = \frac{NDE(T)/T \times Y}{\bar{S}(T)} \times 1000 = \frac{NDE(T) \times Y}{\bar{S}(T) \times T} \times 1000 \quad (1.4)$$

де: K_i^T — оцінка коефіцієнта інтенсивності за період *T* у річному вимірі, %;

NDE(T) — кількість демографічних подій за період *T*;

T — тривалість періоду, *днів*;

Y — кількість днів у році (*Y=365, 366*);

$\bar{S}(T)$ — середня чисельність населення за період *T, осіб*.

У чисельнику формули (1.4) спочатку розраховується число подій за один день (число подій за період ділиться на тривалість періоду у днях), потім оцінюється число подій за рік (число подій за день множиться на кількість днів у році: на 365 у звичайному році або на 366 у високосному). У знаменнику використовується значення середньої чисельності населення за період розрахунку.

Коефіцієнт, розрахований за формулою (1.4), інтерпретується так: яким би міг бути річний коефіцієнт, якби демографічні процеси протягом року відбувались так, як і в періоді, для якого цей коефіцієнт розрахований.

Загальний коефіцієнт народжуваності розраховується за формулою:

$$b = \frac{B}{\bar{S}} \times 1000 \quad (1.5)$$

де:

b — загальний коефіцієнт народжуваності, %;

B — кількість народжених дітей, *осіб*;

$\bar{S}$ — середньорічна чисельність населення, *осіб*.

1.3.2 Розрахунок аналізу смертності

Середній вік померлих обчислюється як середня арифметична зважена віку померлих до кількості померлих відповідного віку:

$$\bar{x} = \frac{\sum x' M_x}{\sum M_x} \quad (1.6)$$

де: $\bar{x}$ — середній вік померлих, *років*;

x' — середина вікового інтервалу, *років*;

M_x — кількість померлих у віці *x, осіб*;

$\sum M_x$ — загальна кількість померлих, *осіб*.

1.3.3 Природний приріст населення

Абсолютний природний приріст (скорочення) населення – різниця між кількістю народжених живих і кількістю померлих:

$$P = B - M \text{ (1.7)}$$

P — природний приріст (скорочення) населення, *осіб*;

B — кількість живо народжених, *осіб*;

M — кількість померлих, *осіб.*

Загальний коефіцієнт природного приросту (скорочення) населення розраховується, як відношення абсолютної величини природного приросту (скорочення) населення до середньорічної чисельності наявного населення:

$$p = \frac{P}{\bar{S}} \times 1000 \text{(1.8)}$$

p — загальний коефіцієнт природного приросту (скорочення) населення, *%*;

P — абсолютна величина природного приросту (скорочення) населення, *осіб*;

$\bar{S}$ — середньорічна чисельність наявного населення, *осіб*.

Для збалансування показників і вихідних даних, отримане в ході розрахунку значення округлюється до десяткового знаку.

Для оцінки ступеня заміщення померлого населення народженими використовується коефіцієнт Покровського (коефіцієнт життєвості):

$$K = \frac{B}{M} \times 100 \text{ (1.9)}$$

де: *K* — коефіцієнт Покровського (коефіцієнт життєвості), *на 100 померлих*;

B — загальна кількість живо народжених протягом року, *осіб*;

M — загальна кількість померлих за рік, *осіб*.

1.3.4 Розрахунок коефіцієнтів шлюбності та розлучуваності

Середній вік при укладені шлюбу обчислюється як середня арифметична зважена віку на кількість укладених шлюбів відповідного віку:

$$\bar{x} = \frac{\sum x' C_x}{\sum C_x} \text{ (1.10)}$$

$\bar{x}$ — середній вік при укладені шлюбу, *років*;

x' — середина вікового інтервалу укладення шлюбу, *років*;

C_x — кількість укладених шлюбів у віці *x, одиниці*;

$\sum C_x$ — загальна кількість укладених шлюбів, *одиниць*.

1.3.5 Загальний коефіцієнт шлюбності

Загальний коефіцієнт шлюбності показує, яка кількість шлюбів припадає на 1000 осіб середньорічної чисельності наявного населення:

$$c = \frac{C}{\bar{S}} \times 1000 \quad (1.11)$$

c — загальний коефіцієнт шлюбності, *%*;

C — кількість шлюбів, *одиниць*;

$\bar{S}$ — середньорічна чисельність наявного населення, *осіб.*

1.3.6 Абсолютні показники розлучення

Розраховується за формулою:

$$D = D_{\text{ОДРФЦС}} + D_{\text{С–СПП}} + D_{\text{С–СОП}} \quad (1.12)$$

D — загальна кількість розлучень, *одиниць*;

$D_{\text{ОДРФЦС}}$ — кількість розірвань шлюбу, зареєстрованих в органах державної реєстрації актів цивільного стану, *одиниць*;

$D_{\text{С–СПП}}$ — кількість ухвалених судами рішень щодо розірвань шлюбу в порядку розгляду справ позовного провадження із задоволенням позову, *одиниць*;

$D_{\text{С–СОП}}$ — кількість ухвалених судами рішень щодо розірвань шлюбу в порядку розгляду справ окремого провадження із задоволенням заяви, *одиниць*.

1.3.7 Загальний коефіцієнт розлучуваності

Загальний коефіцієнт розлучуваності дорівнює відношенню кількості розлучень до середньорічної чисельності наявного населення:

$$d = \frac{D}{\bar{S}} \times 1000 \quad (1.13)$$

d — коефіцієнт розлучуваності, *%;*

D — кількість розлучень, *одиниць;*

$\bar{S}$ — середньорічна чисельність наявного населення, *осіб*.

Інші показники, які впливають на демографічні показники населення України, також розглядаються і зараз проводиться їх аналіз і пошук математичної моделі для їх реалізації.

1.4 Потенційні користувачі

Потенційними користувачами розробленого програмного забезпечення є:

— люди які проводять оцінку національного фонду країни;

— військові;

— державні структури, які складають соціально-політичну оцінку.

2. АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ ІНСТРУМЕНТАЛЬНОГО ЗАСОБУ ДЛЯ ОЦІНКИ СОЦІАЛЬНО-ПОЛІТИЧНИХ РИЗИКІВ З ВИКОРИСТАННЯМ ГІС ТЕХНОЛОГІЙ

«Клієнт-сервер» — обчислювальна або мережева архітектура, в якій завдання або мережеве навантаження розподілені між постачальниками послуг, званими серверами, і замовниками послуг, званими клієнтами. Фактично клієнт і сервер - це програмне забезпечення.

Розглянемо типову архітектуру клієнт-сервер (рисунок 2.1).

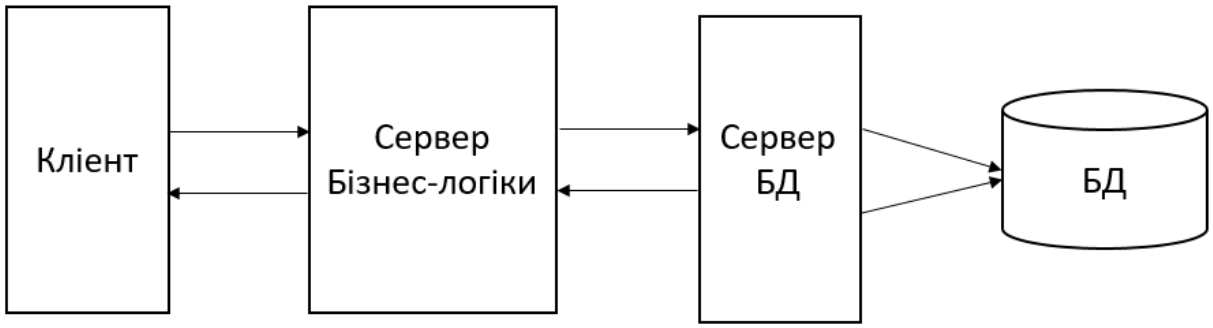


Рисунок 2.1 – Багаторівнева архітектура клієнт-сервер

Кожен клієнт містить лише реалізацію користувацького інтерфейсу та не займається безпосередніми маніпуляціями базою даних. При настанні необхідності виконання певних дій, що потребують втручання у віддалену частину системи, виконується запит до серверу додатків, що містить всю необхідну бізнес-логіку для обробки даних. Сервер додатків в свою чергу делегує виконання безпосередніх операцій над даними в БД серверу баз даних, що займається виконанням необхідних команд над інформацією, яка міститься в базі даних.

Таким чином, кожен рівень відповідає лише за окремий функціонал та делегує виконання повноважень, які йому не належать, іншому рівню. Це надає можливість паралельної розробки кожного рівня, легкого масштабування проекту за необхідністю та окремого тестування кожного з незалежних частин програми.

Для подальшого відображення даних, які знаходяться на рівні бази даних, використовується інший програмній засіб. В ArcGis створюється підключення до сервера бази даних та забезпечується доступ до даних. Після усіх підключень налаштовується відображення інформації для інтуїтивно-логічного розуміння та подальшого використання.

2.1 Опис предметної області

Головним багатством країни є її людський потенціал, який є основою її існування. У першу чергу населення - носій інтелектуального потенціалу значно зростає в постіндустріальних державах, де інтелект нації стає саме головною

рушійною силою і визначальним фактором прогресу. Будь-яка втрата людських ресурсів незалежно від причин і характеру, як в кількісному, так і в якісному аспектах, це не тільки проблема внутрішнього характеру, а й геополітична проблема. Ці втрати дестабілізують і послаблюють країну, стаючи загрозою національній безпеці.

Демографічний фактор є одним з ключових факторів забезпечення стабільного і безпечний розвитку країни, а питання демографічного розвитку слід розглядати як фактор, і в той же час як результат існування держави.

Стан демографічної безпеки є найкращим для суспільства. Категорія безпеки в демографічній системі аналогічна функції надійності в технічній системі. Якщо надійність є узагальненою характеристикою якості технічної системи, то безпека є узагальненою характеристикою якості демографічної системи.

Вирішальним для майбутнього існування демографічних систем - потенціал «зберегти свої системно-формуючі властивості, основні характеристики, параметри та стан патогенних (дезорганізуючих, деструктивних, деструктивних) впливів від різних суб'єктів, явищ або процесів», тобто перебувати в стані безпеки. Це свідчить про порушення функціонування демографічної системи і виникнення небезпеки.

2.2 Висновки до розділу

У даному розділі вказано на проблематику тематики демографічної безпеки та описані її основні складові. Також у цьому розділі розглянуто принцип створення розподілених програмних продуктів за допомогою використання багаторівневої архітектури і наведені основні переваги даного принципу.

3. ЗАСОБИ РОЗРОБКИ

Для вирішення задачі оцінки соціально-політичних ризиків існують повноцінні програмні комплекси: GRASS GIS, Mapline, Maptitude, QGIS та інші[3].

Вище згадані рішення відносяться до ліцензованого програмного забезпечення та мають високу вартість. Придбавши подібне програмне забезпечення користувач все одно не має низки можливостей та отримує обмежений доступ до внутрішнього функціоналу, заборонений доступ до вивчення коду и внесення змін.

Програмний продукт для оцінки соціально-політичних ризиків містить простий та зрозумілий для користувача інтерфейс. Містить тільки необхідний функціонал, який позбавлений надмірної кількості інструментів. Крім того, при використанні даного програмного застосунку завжди є можливість ознайомитись з кодом програми, так як код є відкритим і доступним до вивчення, а саме використання додатку – безкоштовне.

Вище були перелічені переваги користування наданого програмного продукту перед існуючими, масштабними аналогами. Також в даному розділі наведено основні технології, засоби та принципи розробки програмного забезпечення.

3.1 Платформа .Net Framework

Microsoft .NET[4] — це технологія, впроваджена корпорацією Майкрософт як платформа для створення звичайних програм для персональних комп'ютерів, а також веб-додатків або порталів. Однією з ідей .NET є створення сумісності служб, написаних на різних мовах програмування. Ця функція впроваджується корпорацією Майкрософт як перевага для .NET

3.1.1 Загальні відомості про платформу .NET Framework

Платформа .NET Framework — це технологія, яка підтримує створення та виконання XML-додатків і веб-служб. При розробці .NET Framework основними завданнями були:

— забезпечення узгодженого об’єктно-орієнтованого середовища програмування для локального збереження і виконання об’єктного коду, для виконання коду, розподіленого між різними серверами, пов’язаними один з одним мережею Internet;

— середовище забезпечення, яке мінімізує конфлікти при розгортанні програмного забезпечення та керуванням версіями;

— забезпечення середовища продуктивності, що забезпечує безпечне виконання коду, включаючи код, створений невідомим або ненадійним виробником третьої сторони;

— забезпечення середовища виконання коду, що виключає проблеми з виконанням середовищ виконання сценаріїв або інтерпретацією коду;

— забезпечення уніфікованих принципів розробки для різних типів програм, таких як програми Windows і веб-додатки;

— розширене співробітництво на основі галузевих стандартів, які інтегрують код .NET Framework з будь-яким іншим кодом.

Платформа .NET Framework складається з загальних середовищ виконання (середовища CLR) і бібліотек класів .NET Framework. Основою для .NET Framework є середовище CLR. Середовище виконання може розглядатися як агент, який керує кодом під час виконання програми і надає основні послуги, такі як управління пам'яттю, управління асинхронними потоками і віддалену взаємодію. У цьому

випадку створюються умови суворої типізації та інших видів контролю правильності коду, які забезпечують безпеку і надійність при роботі. Фактично основним завданням середовища виконання є управління кодом. Код, який звертається до середовища виконання, називається керованим кодом, а код, який не має доступу до середовища виконання, називається некерованим кодом. Бібліотека класів - це повна об'єктно-орієнтована колекція, що дозволяє повторно використовувати типи, що використовуються для розробки програм — від звичайних програм, що виконуються в консолі за допомогою командного рядка або графічного інтерфейсу користувача (GUI), до програм, використовуючи найновішу ASP. Технології NET Core.

Платформа .NET Framework може розміщуватися на некерованих компонентах, які підключають середовище CLR до власних процесів і виконують виконання керованого коду, створюючи тим самим програмну оболонку, яка дозволяє використовувати як керований, так і некерований код. Платформа .NET Framework не тільки забезпечує кілька основних середовищ виконання, але й підтримує розробку основних середовищ, що пропонуються сторонніми постачальниками.

Наприклад, ASP.NET Core розміщує середовище виконання й надає масштабоване середовище для керованого коду на стороні сервера. ASP.NET Core працює безпосередньо з середовищем виконання, щоб забезпечити виконання програм.

Браузер Internet Explorer може слугувати прикладом некерованої програми, яка розміщує середовище виконання (як розширення типу MIME). Реалізація середовищ в Internet Explorer дозволяє реалізувати керовані компоненти або елементи керування Windows Forms в документах HTML. Це розміщення середовища дозволяє виконувати керований мобільний код і скористатися ним, включаючи виконання в умовах неповної довіри та ізоляції файлів.

На малюнку 3.1 показано взаємозв'язок між середовищем CLR і бібліотекою класів з однією програмою і всією системою. На малюнку 3.1 також показано, як керований код працює в більш широкій архітектурі.

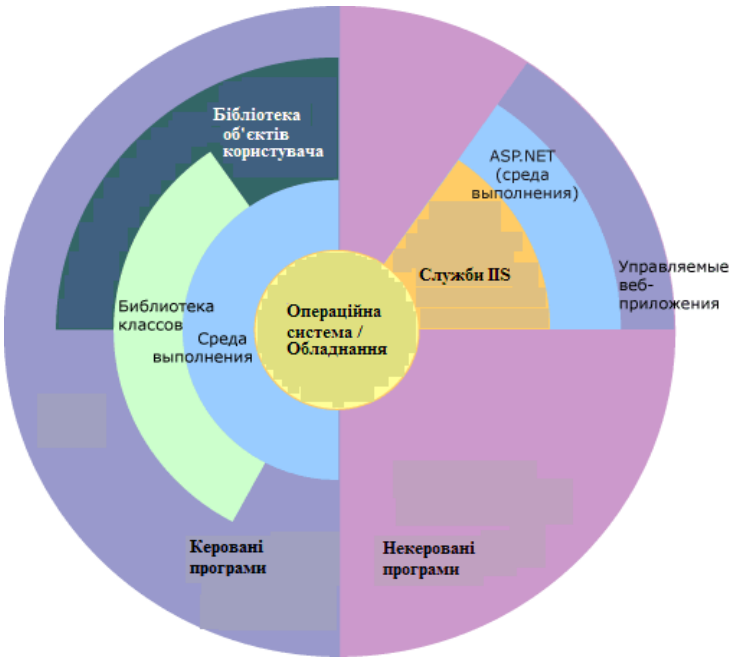


Рисунок 3.1 – Модель виконання .Net Framework

3.1.2 Середовище CLR

Середовище CLR[5] контролює пам'ять, робочий процес, виконання коду, перевірку безпеки коду, компіляцію проекту та різні системні процеси (рисунок 3.2). Ці процеси є внутрішніми для керованого коду, які виконується за допомогою CLR.

З точки зору безпеки, керованим компонентам призначаються різні ступені довіри залежно від ряду факторів, включаючи їх походження (наприклад, Інтернет, корпоративну мережу або локальний комп'ютер). Це вказує, що керований компонент може або не може виконувати операції доступу до файлів, операції доступу до реєстру або інші важливі функції, навіть якщо він використовується в одному активному додатку.

Середовище реалізації забезпечує контроль доступу для коду. Наприклад, користувачі можуть довірити виконувану програму, реалізовану на веб-сторінці, відтворення анімації на екрані або звукозапису, не дозволяючи їм отримувати доступ до своїх особистих даних, файлової системи або мережі. Таким чином, засоби захисту CLR забезпечують, дійсно, широкі можливості з надзвичайно багатими функціями.

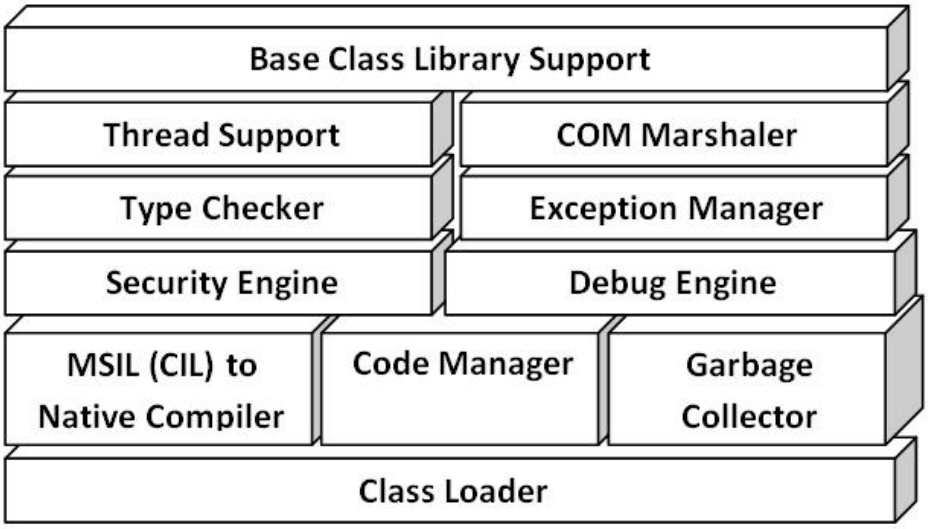


Рисунок 3.2 – Архітектура CLR

Середовище реалізації також забезпечує надійність коду шляхом впровадження суворої типографічної інфраструктури та перевірки коду, що називається системою загального типу (CTS). Система загальних типів забезпечує запис всього керованого коду. Різні компілятори від Microsoft і незалежних виробників створюють керований код, який задовольняє систему загальних типів. Це означає, що керований код може приймати інші керовані типи і екземпляри, забезпечуючи при цьому правильність типів і суворе введення.

Крім того, кероване середовище виконання виключає багато рішень частих проблем програмного забезпечення. Наприклад, середовище виконання автоматично керує розміщенням об'єктів і посилань на об'єкти, звільняючи їх, коли вони більше не використовуються. Автоматичне керування пам'яттю усуває дві найпоширеніші помилки програми: втрата пам'яті та недійсні посилання на об’єкти.

Робоче середовище також підвищує продуктивність розробників. Для прикладу, програмісти можуть писати програми на звичайній мові програмування, використовуючи середовище виконання, бібліотеку класів і компоненти, написані іншими розробниками на інших мовах. Це доступне кожному виробнику компіляторів, які мають доступ до середовища виконання. Компілятори, призначені для .NET Framework, роблять .NET Framework доступним для існуючого коду, написаного на відповідних мовах, що значно полегшує процес передачі існуючих програм.

Хоча середовище впровадження було розроблено для майбутнього програмного забезпечення, воно також підтримує сучасне та старе програмне забезпечення. Взаємодія керованих і некерованих кодів дозволяє розробникам використовувати необхідні компоненти COM і DLL.

Середовище впровадження призначене для підвищення продуктивності. Незважаючи на те, що середовище виконання передбачає багато стандартні служби часу виконання, керований код ніколи не інтерпретується. Інструмент компіляції за запитом (JIT) дозволяє виконувати всі керовані коди на машинній мові. Тим часом менеджер пам'яті усуває можливість фрагментації пам'яті і збільшує обсяг пам'яті для додаткового підвищення продуктивності.

Нарешті, середовище виконання може бути розміщено у високопродуктивних серверних програмах, таких як Microsoft SQL Server і IIS (Internet Information Services). Ця інфраструктура дозволяє використовувати керований код для написання власної логіки програми, використовуючи високу продуктивність найкращих виробничих серверів, що підтримують середовище виконання.

3.2 Мова програмування C#

Мова C#[6,7] — це чергова сходинка нескінченної еволюції мов програмування. Її створення викликано процесом вдосконалення і адаптації, який визначав розробку комп'ютерних мов протягом останніх років. Подібно до всіх успішних мов, які побачили світ раніше, C# спирається на минулі досягнення мистецтва програмування, яке постійно розвивається.

У мові C # (створеному компанією Microsoft для підтримки середовища .NET Framework) перевірені часом засоби вдосконалені за допомогою найсучасніших технологій. C# надає дуже зручний і ефективний спосіб написання програм для сучасного безпечного середовища обчислювальної обробки даних, яка включає операційну систему Windows, Internet та інші компоненти. В процесі становлення мові C# "дісталася" багата спадщина. Вона — прямий нащадок двох найбільш успішних мов програмування (C і C++). Розуміння природи цих взаємозв'язків вкрай

важливе для розуміння C#. C# містить безліч нових засобів, найважливіші з них пов'язані з вбудованою підтримкою програмних компонентів. Саме наявність вбудованих засобів написання програмних компонентів і дозволило C# називатися компонентно-орієнтованою мовою. Наприклад, C# включає засоби, які безпосередньо підтримують складові частини компонентів: властивості, методи і події. Все ж найважливішою якістю компонентно-орієнтованої мови є її здатність працювати в середовищі багатомовного програмування.

Незважаючи на те, що C # — самодостатня комп'ютерна мова, у неї особливі взаємозв’язки з середовищем .NET Framework. І на це є дві причини. По-перше, C# спочатку був розроблений компанією Microsoft для створення коду, що виконується в середовищі .NET Framework. По-друге, в цьому середовищі визначені бібліотеки, використовувані мовою C#. І хоча можна відокремити C# від .NET Framework, ці два середовища тісно пов'язані, тому дуже важливо мати загальне уявлення про .NET Framework і розуміти, чому це середовище настільки важливе для C#.

Оболонка .NET Framework визначає середовище для розробки і виконання сильно розподілених додатків, заснованих на використанні компонентних об'єктів. Вона дозволяє "мирно співіснувати" різним мовам програмування і забезпечує безпеку, переносимість програм і загальну модель програмування для платформи Windows. Важливо при цьому розуміти, що .NET Framework за своєю сутністю не обмежена застосуванням в Windows, тобто програми, написані для неї, можна потім переносити в середовища, відмінні від Windows. Зв'язок середовища .NET Framework з C# обумовлений наявністю двох дуже важливих засобів.

Одне з них, Common Language Runtime (CLR), являє собою систему, яка управляє виконанням призначених для користувача програм. CLR — це складова частина .NET Framework, яка робить програми багатоплатформними, підтримує багатомовне програмування і забезпечує безпеку. Другий засіб, бібліотека класів .NET-оболонки, надає програмам доступ до середовища виконання. Наприклад, якщо вам потрібно виконати операцію вводу-виводу: відобразити що-небудь на екрані, то для цього необхідно використовувати .NET. Якщо програма обмежується використанням засобів, визначених .NET-бібліотекою класів, вона може

виконуватися всюди (тобто в будь-якому середовищі), де підтримується .NET-система. Оскільки C# автоматично використовує .NET-бібліотеку класів, C# програми автоматично переносяться в усі .NET-середовища. Система CLR управляє виконанням .NET-коду. Ось, як це відбувається. Внаслідок компіляції C# програми отримуєтеся не виконуваний код, а файл, який містить спеціальний псевдокод, іменований проміжним мовою Microsoft (Microsoft Intermediate Language - MSIL). MSIL визначає набір інструкцій, які не залежать від типу процесора. Мета CLR-системи - при виконанні програми перевести її проміжний код в виконуваний. Таким чином, програма, піддана MSIL-компіляції, може бути виконана в будь-якому середовищі, для якої реалізована CLR-система. У цьому частково і складається здатність середовища .NET Framework домагатися переносимості програм.

Код, написаний проміжною мовою Microsoft, перекладається в виконуваний за допомогою JIT — компілятора. "JIT" - скорочено від виразу "just-in-time", що означає виконання точно до потрібного моменту (так позначається стратегія прийняття рішень в найостанніший відповідний для цього момент з метою забезпечення їх максимальної точності). Цей процес працює наступним чином. При виконанні .NET-програми CLR-система активізує JIT-компілятор, який перетворює MSIL-код в її "рідний" код на необхідній основі, оскільки необхідно зберегти кожну частину програми.

Таким чином, C# програма виконується у вигляді "рідного" коду, незважаючи на те, що спочатку вона була скомпільована в MSIL-код. Це означає, що програма буде виконана практично так само швидко, як коли б вона з самого початку була скомпільована з отриманням "рідного" коду, але з "додаванням" переваг переносимості від перетворення в MSIL-код.

В результаті компіляції співпрограми крім MSIL-коду утворюються і метадані (metadata). Вони описують дані, які використовуються програмою, і дозволяють коду взаємодіяти з іншим кодом. Метадані містяться в тому ж файлі, де зберігається MSIL-код.

У загальному випадку при написанні співпрограми створюється код, званий керованим (managed code). Керований код виконується під управлінням CLR-

системи. У такого виконання в результаті є як певні обмеження, так і чималі переваги. До обмежень відноситься необхідність мати спеціальний компілятор, який повинен створювати MSIL-файл, призначений для роботи під управлінням CLR-системи, а також, цей компілятор повинен використовувати бібліотеки середовища .NET Framework. Переваги ж керованого коду — сучасні методи управління пам'яттю, можливість використовувати різні мови, поліпшена безпека, підтримка управління версіями і чітка організація взаємодії програмних компонентів. Всі Windows-програми до створення середовища .NET Framework використовували некерований код, який не виконується CLR-системою. Керований і некерований код можуть працювати разом, тому факт створення C# компілятором керованого коду аж ніяк не обмежує його можливість виконуватися спільно з раніше створеними програмами.

Незважаючи на те, що керований код володіє достоїнствами, наданими CLR-системою, але якщо він використовується іншими програмами, написаними на інших мовах, то для досягнення максимальної зручності і простоти використання він повинен відповідати специфікації універсальної мови (Common Language Specification — CLS). Ця специфікація описує набір властивостей, які одночасно повинні володіти різними мовами. Відповідність CLS-специфікації особливо важливо при створенні програмних компонентів, які призначені для використання програмами, написаними на інших мовах. CLS-специфікація включає підмножину систем підтримки загальних типів (Common Type System - CTS).

3.3 Плаский дизайн

Плаский дизайн — дизайн інтерфейсів програм і операційних систем, представлений, як протилежність реалізму. За задумом «плаский дизайн» повинен підкреслювати ефект «чарівної простоти» і витонченості.

Почав стрімко набирати популярність і зараз стає новим стандартом в дизайнерському комп'ютерному напрямку.

Дизайни в цьому стилі включають в себе дуже мало деталей, таких як текстури і контейнери. Ці макети використовують чисті кольори, уникають градієнтів і використовують більше квадратних форм з гострими кутами.

Пласкі конструкції складаються з мінімального проектування, створюючи чисту і пласку повну картинку, що допомагає сформувати користувальницький інтерфейс впорядкованішим і більш інформативним.

Це не тільки новий напрям в дизайні для користувача інтерфейсу. Цей дизайн охоплює ідеологію "менше означає більше". Однак і простоту не завжди дуже легко побудувати.

Щоб створити плаский, спеціально розроблений інтерфейс, необхідно тримати всі елементи відносно простими, зробити мінімум можливих елементів, зберігаючи при цьому функціональність пристрою як першочерговий пріорітет. Необхідно дотримуватися приємної та легкої колірної гами, яка не повинна відволікати від інформації, що використовує користувач (рисунок 3.3).

Рисунок 3.3 – приклад плаского дизайну

Пласка конструкція вимагає мінімального використання елементів, набагато менше градієнтів і об'ємного блиску для іконок, що дозволяє користувачам швидко переміщуватися по екрану очима, не відволікаючись від змісту сторінки.

3.4 Середовище розробки Visual Studio

Microsoft Visual Studio[8] — один з продуктів компанії Майкрософт, який представляє з себе інтегровану середу розробки програмного забезпечення і ряд інших інструментальних засобів. Даний продукт дозволяє розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в не керованому, так і керованому кодах для всіх платформ, підтримуваних Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework і Microsoft Silverlight.

Visual Studio включає в себе редактор вихідного коду з підтримкою технології IntelliSense і можливістю найпростішого рефакторингу коду. Вбудований відладчик може працювати як відладчик рівня вихідного коду, так і як відладчик машинного рівня. Решта вбудовуваних інструментів включають в себе редактор форм для спрощення створення графічного інтерфейсу додатку, веб-редактор, дизайнер класів і дизайнер схеми бази даних. Visual Studio дозволяє створювати і підключати сторонні додатки (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду, додавання нових наборів інструментів, наприклад, для редагування і візуального проектування коду на предметно-орієнтованих мовах програмування або інструментів для інших аспектів циклу розробки програмного забезпечення. Загальний вигляд вікна, яке відображається при завантаженні Visual Studio, початок роботи та створення нового проекту (рисунок 3.4)

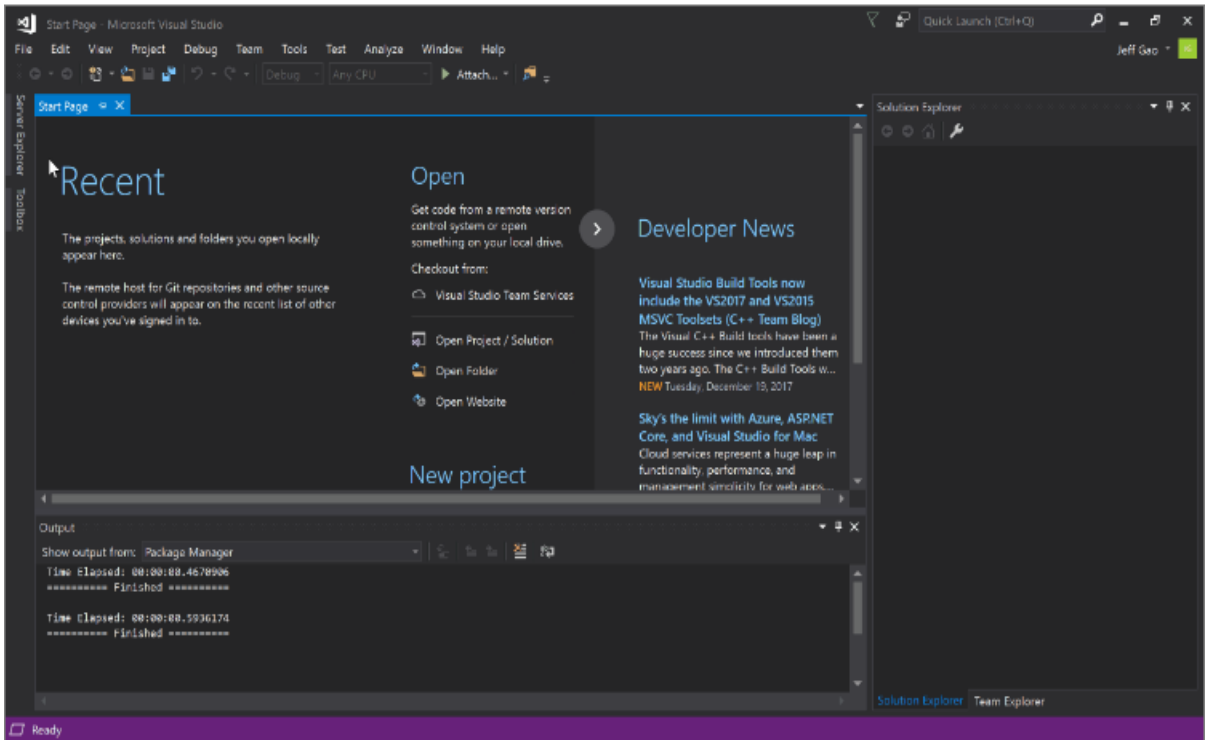


Рисунок 3.4 — Visual Studio

3.5 Технологія для розробки інтерфейсу

Для створення клієнтської частини порталу були використані такі технології як Windows Forms[9]. Windows Forms впроваджує засоби для:

— створення структурованого інтерфейсу користувача;

— отримання інформації з форми;

— створення інтерактивних форм.

Windows Forms — інтерфейс програмування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код - класи, що реалізують API для Windows Forms, що не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні ПЗ на C #, C ++, так і на VB.Net, J # і ін.

З одного боку, Windows Forms розглядається як заміна більш старої і складної бібліотеки MFC, спочатку написаної на мові C ++. З іншого боку, WF не пропонує парадигму, порівняно з MVC. Для виправлення цієї ситуації і реалізації даної

функціональності в WF існують сторонні бібліотеки. Однією з найбільш використовуваних подібних бібліотек є User Interface Process Application Block, випущена спеціальною групою Microsoft, що займається прикладами і рекомендаціями, для безкоштовного скачування. Ця бібліотека також містить вихідний код і навчальні приклади для прискорення навчання.

3.6 Бібліотека Entity framework

Entity Framework[10] являє собою спеціальну об'єктно-орієнтовану технологію на базі фреймворка .NET для роботи з даними. На відміну від засобів ADO.NET які дозволяють створювати підключення, команди та інші об'єкти для взаємодії з базами даних, то Entity Framework являє собою більш високий рівень абстракції, який дозволяє абстрагуватися від самої бази даних і працювати з даними незалежно від типу сховища. Якщо на фізичному рівні ми оперуємо таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує Entity Framework, ми вже працюємо з об'єктами мови C#.

Центральною концепцією Entity Framework є поняття сутності або entity. Сутність представляє набір даних, асоційованих з певним об'єктом. Тому дана технологія передбачає роботу не з таблицями, а з об'єктами і їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має низку властивостей. Наприклад, якщо сутність описує людину, то ми можемо виділити такі властивості, як ім'я, прізвище, зріст, вік, вага. Властивості необов'язково представляють прості дані типу int, а й можуть представляти більш комплексні структури даних. І у кожної сутності може бути одна або кілька властивостей, які будуть відрізняти цю сутність від інших і будуть унікально визначати цю сутність. Подібні властивості називають ключами.

При цьому суті можуть бути пов'язані асоціативним зв'язком один-до-багатьох, один-до-одного і багато-до-багатьох, подібно до того, як в реальній базі даних відбувається зв'язок через зовнішні ключі.

Відмінною рисою Entity Framework є використання запитів LINQ для вибірки даних з БД. За допомогою LINQ ми можемо не тільки отримувати певні рядки, що зберігають об'єкти, з бази даних, а й отримувати об'єкти, пов'язані різними асоціативними зв'язками.

Іншим ключовим поняттям є Entity Data Model. Ця модель зіставляє класи сутностей з реальними таблицями в БД.

Entity Data Model складається з трьох рівнів: концептуального, рівень сховища і рівень зіставлення.

На концептуальному рівні відбувається визначення класів сутностей, які використовуються в додатку.

Рівень сховища визначає таблиці, стовпці, відносини між таблицями і типи даних, з якими порівнюється використовувана база даних.

Рівень зіставлення служить посередником між попередніми двома, визначаючи зіставлення між властивостями класу суті і стовпцями таблиць.

Таким чином, ми можемо через класи, визначені у додатку, взаємодіяти з таблицями з бази даних.

Способи взаємодії з БД. Entity Framework передбачає три можливі способи взаємодії з базою даних:

— Database first: Entity Framework створює набір класів, які відображають модель конкретної бази даних

— Model first: спочатку розробник створює модель бази даних, по якій потім Entity Framework створює реальну базу даних на сервері.

— Code first: розробник створює клас моделі даних, які будуть зберігатися в бд, а потім Entity Framework за цією моделлю генерує базу даних і її таблиці.

Для використання обраного фреймворку потрібно завантажити додаткові бібліотеки, це можна зробити за допомогою NuGet Package Manager (рисунок 3.6).

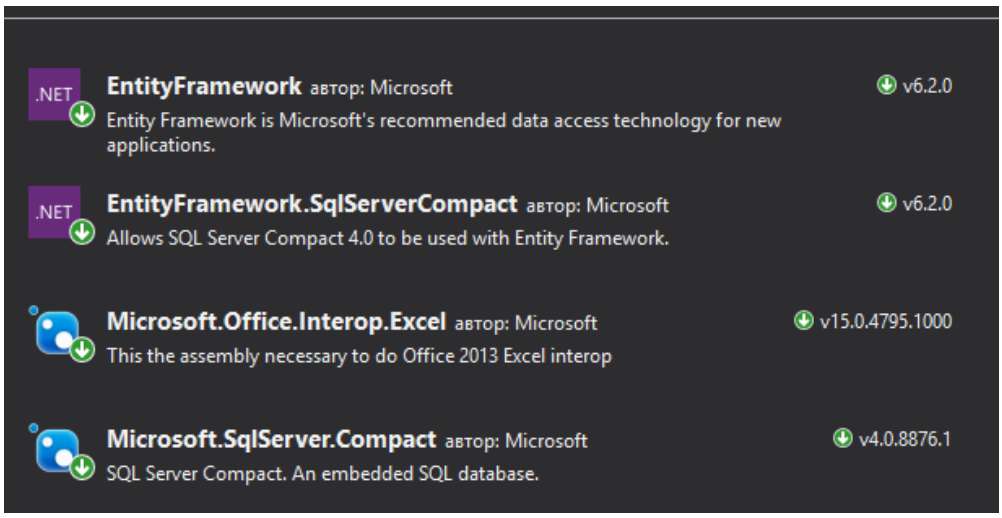


Рисунок 3.6 — NuGet Package Manager

NuGet — система управління пакетами для платформ розробки Microsoft, в першу чергу бібліотек .NET Framework.

3.7 Середовище Microsoft SQL Server

Microsoft SQL Server[11] — це система управління базами даних, запропонована Microsoft. Мова, що використовується для написання SQL-запитів - Transact-SQL, створений спільно Microsoft і Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO для структурованої мови запитів SQL з розширеннями. Він використовується як для малих, так і середніх баз даних, а також для великих корпоративних баз даних. Протягом багатьох років вона успішно конкурує з іншими системами управління базами даних.

Transact-SQL є реалізацією SQL-92 (стандарт ISO для SQL) з багатьма розширеннями. T-SQL дозволяє використовувати додатковий синтаксис збережених процедур і забезпечує підтримку транзакцій (взаємодія з базою даних з менеджером додатків). Microsoft SQL Server та Sybase ASE для мережевих інтерфейсів використовують протокол прикладного рівня під назвою Табличний потік даних (TDS, табличний протокол передачі даних).

Microsoft SQL Server також підтримує Open Database Connectivity (ODBC), інтерфейс для взаємодії з додатками СУБД. SQL Server надає можливість підключення користувачів через сервіси веб-серверів, які використовують протокол SOAP. Це дозволяє клієнтським програмам, які не призначені для крос-платформних Windows, підключатися до SQL Server.

Корпорація Майкрософт також випустила сертифікований драйвер JDBC, що дозволяє підключати Java-програми (такі як BEA та IBM Websphere) до Microsoft SQL Server.

SQL Server підтримує дзеркальні та кластерні бази даних. Кластер SQL Server - це набір ідентичних серверів; Ця схема допомагає розподілити навантаження між декількома серверами. Всі сервери мають одне віртуальне ім'я, а дані розподіляються відповідно до IP-адрес кластерних машин протягом робочого циклу. Крім того, у разі відмови або відмови на одному з кластерних серверів можливе автоматичне перенесення завантаження на інший сервер.

SQL Server підтримує надмірне дублювання даних у трьох сценаріях:
Знімок: Здійснюється знімок бази даних, яку сервер надсилає одержувачам.
Історія змін: всі зміни бази даних постійно передаються користувачам.
Синхронізація з іншими серверами: Бази даних декількох серверів синхронізуються один з одним. Зміни до всіх баз даних відбуваються незалежно на кожному сервері, а синхронізація даних відбувається під час синхронізації. Дублювання цього типу передбачає можливість вирішення конфліктів між базами даних.

SQL Server має вбудовану підтримку .NET Framework. Завдяки цьому, збережені процедури бази даних можуть бути написані на будь-якій мові платформи .NET, використовуючи повний набір бібліотек, доступних для .NET Framework. На відміну від інших процесів, .NET Framework виділяє додаткову пам'ять і створює елементи керування SQL Server без використання вбудованих інструментів Windows. Це покращує продуктивність за звичайних алгоритмів Windows, оскільки алгоритми розподілу ресурсів спеціально налаштовані для використання в структурах SQL Server.

3.8 Середовище ArcGIS

Програмний продукт ArcGIS[12] компанії Esri є програмним забезпеченням ГІС (географічних інформаційних систем), який дозволяє користувачам керувати та аналізувати географічну інформацію шляхом візуалізації діапазону географічних статистичних даних через карти та побудову шарів (наприклад, торгові потоки або кліматичні дані). Він використовується цілим рядом академічних інститутів і департаментів, як у науці, так і в гуманітарних науках, для розробки та ілюстрування новаторських досліджень. Далі ця технологія використовується багатьма урядовими організаціями та комерційними чи приватними установами по всьому світу.

Система має можливість зробити географічну інформацію доступною по всьому інституту, компанії, приватною або відкритою в Інтернеті. Таким чином, програмне забезпечення, по суті, працює як платформа, через яку географічні дані можуть бути з'єднані, розподілені та проаналізовані.

Серед останніх інноваційних аспектів інтеграції платформ, користувачі можуть включати в себе 3D-моделювання, просторовий аналіз, комплексну візуалізацію карти, навігацію даними і географічний збір інформації в реальному часі, все через засоби управління даними та збору даних. Таким чином, коли попередні версії програми обмежувалися географічним відображенням інформації, вона перетворилася на провідну платформу з найсучаснішими можливостями хмарних технологій.

Як і більшість програм ГІС, ArcGIS створює карти, які вимагають об’єкти, організовані як шари. Кожен шар реєструється просторово, так що, коли вони накладаються один на інший, програма виводить їх правильно, щоб побудувати складну карту даних. Базовий шар майже завжди є географічною картою, витягнутою з ряду джерел залежно від необхідної візуалізації (дорожня карта, супутник тощо). Ця програма має багато з цих доступних елементів для користувачів і навіть включає живі приклади шарів, включаючи інформацію про дорожній рух.

Дані можуть бути пов'язані з будь-яким з цих просторових шарів і можуть бути як відображені, так і проаналізовані, чи то через атрибути, такі як демографічні зміни, чи теж через таблиці даних.

Однак те, що відрізняє цю систему від своїх конкурентів, є складною платформою, за допомогою якої ці дані і відображення можуть бути виконані. Таким чином, це дуже багатофункціональна програма, що підлягає новим оновленням і вдосконаленням. В даний час вона доступна лише на робочих машинах з операційною системою Microsoft Windows, хоча онлайн-програма доступна на більшості операційних систем. Оскільки вона функціонує як веб сервіс. Підсумовуючи, це єдине рішення для збору даних, управління та аналізу як фільтрування через створення карт.

Це програмне забезпечення дозволяє швидко створювати приголомшливі візуальні карти і моделі, включаючи 3D візуалізації та карти потоків населення. Завдяки функції перетягування, електронні таблиці даних можуть швидко завантажуватися на хмарний сервіс та візуалізуватися. Існує також інтелектуальний інструмент зіставлення, який пропонує найкращі класифікації, стилі та кольори, які відповідають вашим даним.

Зображення доступні у високій якості, взятому як з історичних, так і недавніх джерел у всьому світі, що дозволяє будувати історичні карти, а також останні спостереження за демографічними даними. Поверхневі явища, такі як температура, висота, кількість опадів тощо, також можуть бути повністю інтегровані в ці візуальні карти і моделі з унікальними інструментами для аналізу поверхні.

Як провідна платформа, набір інструментів і додатків, які є центральними для цієї програми, використовується більшістю установ, компаній і відділів, що займаються аналізом географічних даних. Проте завдяки простоті його інтерфейсу в Інтернеті також спостерігається прискорення цінності журналістського та медіа-використання.

Програмне забезпечення Esri має потужну історію та репутацію. Цей факт робить його основним елементом програмного забезпечення для багатьох компаній, що займаються географічними інформаційними системами. Зокрема, вона

використовується місцевими та державними урядами по всьому світу, у тому числі в Сполучених Штатах Америки.

Це програмне забезпечення поставляється в декількох різних варіаціях, від стандартного настільного пакета до виключно веб-програми. Настільний пакет включає в себе базовий набір для публікації та управління даними, а також надання доступу як до онлайнових, так і до "Enterprise" варіантів. Онлайн-версія містить багато функцій, необхідних для створення веб-карт і веб-додатків з використанням географічних даних. Існує готова галерея базових карт і стилів для вибору, а також ціла низка наборів даних для візуалізації.

Переваги он-лайнової програми включають обмін вмістом по всій вашій організації і навіть поза її межами. Групи можуть отримати доступ до приватних карт на запрошення, що дозволяє співпрацювати. Додаткові компоненти програмної платформи включають в себе програми, такі як засоби збору, навігації та геодезії. Більше того, пробні облікові записи можуть бути зареєстровані для ознайомлення з програмним забезпеченням, і вони дають змогу перевірити, чи відповідає він вашим вимогам.

3.9 Висновки до розділу

У цьому розділі розглянуті обрані інструменті та технології, які були необхідні для створення програмного продукту. Розроблений програмний продукт написано об’єктно-орієнтованою мовою програмування C# з використанням розширення WinForms платформи .NET Framework. Розробка проводилась з використанням середовища Microsoft Visual Studio та за допомогою принципів плаского дизайну створення графічних інтерфейсів. Для роботи за базою даних було обрано та використано Microsoft SQL Server та бібліотеку Entity Framework. Відображення даних на карті відбувається за допомогою програмного комплексу ArcGIS.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Розроблений програмний продукт дозволяє користувачу завантажити з комп’ютера користувача файл, який містить статистичні дані демографічної безпеки України та записати отримані дані до бази даних. Після заповнення бази даних користувач має можливість відобразити введені дані на мапі України.

4.1 Архітектура додатку для роботи з БД

Для роботи з серверною частиною використовується принцип "Onion Architecture".

Більшість традиційних архітектур піднімають фундаментальні питання жорсткого зв'язку та розділення проблем. Архітектура Onion була представлена Джеффрі Палермо, щоб забезпечити кращий спосіб побудови додатків у перспективі кращої тестованості, ремонтопридатності та надійності. Onion Architecture вирішує проблеми, з якими стикаються трирівневі та n-ярусні архітектури, а також забезпечують вирішення загальних проблем. Схеми архітектури цибулі взаємодіють між собою за допомогою інтерфейсів.

Onion-архітектура базується на принципі інверсії управління. Цибульна архітектура складається з декількох концентричних шарів, що взаємодіють один з одним по відношенню до ядра, що представляє домен. Архітектура не залежить від

рівня даних, як у класичних багаторівневих архітектурах, а від реальних моделей доменів.

Згідно традиційної архітектури, інтерфейс UI взаємодіє з бізнес-логікою, а бізнес-логіка - до шару даних, всі шари змішуються і сильно залежать один від одного. У 3-ярусної і n-ярусної архітектури жоден з шарів не є незалежними; цей факт викликає розмежування проблем. Такі системи дуже важко зрозуміти і підтримувати. Недоліком цієї традиційної архітектури є непотрібне зв'язування.

Onion-архітектура[13] це поділ програми на рівні. При чому є один незалежний рівень, який знаходиться в центрі архітектури. Від цього рівня залежить другий рівень, від другого - третій і так далі. Тобто виходить, що навколо першого незалежного рівня нашаровується другий-залежний. Навколо другого нашаровується третій, який також може залежати і від першого. Образно це може бути виражено у вигляді цибулини, в якому також є серцевина, навколо якого нашаровуються всі інші верстви, аж до лушпиння (рисунок 5.2).

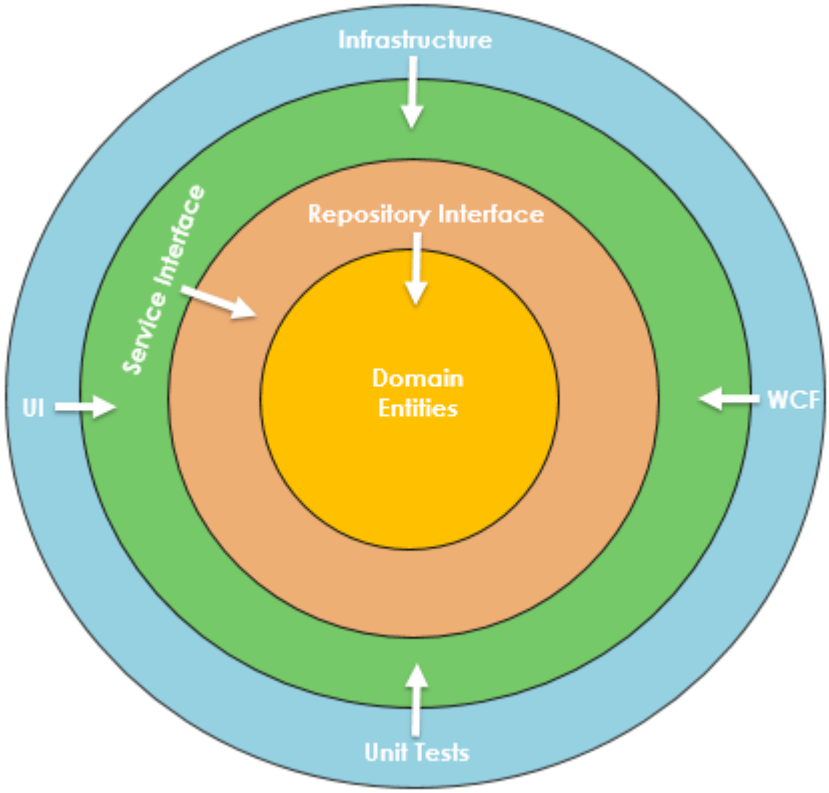


Рисунок 3.5 — Onion-архітектура

Onion-архітектура вирішила ці проблеми шляхом визначення шарів від ядра до інфраструктури. Він застосовує фундаментальне правило, переміщаючи всі зв'язки до центру. Ця архітектура, безсумнівно, зміщена до об'єктно-орієнтованого програмування, і вона ставить об'єкти перед усіма іншими. У центрі цибульної архітектури знаходиться доменна модель, що представляє собою бізнес і об'єкти поведінки. Навколо шару домену є інші шари з більшою кількістю поведінки.

Цифрова архітектура використовує концепцію шарів, але вони відрізняються від 3-х рівневих і n-ярусних архітектурних шарів.

У центральній частині архітектури Onion існує шар домену; цей шар представляє об'єкти бізнесу та поведінки. Ідея полягає в тому, щоб всі об'єкти вашого домену були в цьому ядрі. Він містить всі об'єкти домену додатків. Крім предметів домену, ви також можете мати інтерфейси домену. Ці об'єкти домену не мають залежностей. Об'єкти домену також є рівними, як вони повинні бути, без будь-якого важкого коду або залежностей.

Шар сховища створює абстракцію між суб'єктами домену та бізнес-логікою програми. У цьому шарі ми зазвичай додаємо інтерфейси, які забезпечують збереження та отримання об'єктів зазвичай за допомогою бази даних. Цей шар складається з шаблону доступу до даних, який є більш вільно підключеним підходом до доступу до даних. Ми також створюємо загальний репозиторій і додаємо запити для отримання даних з джерела, зіставляємо дані з джерела даних до суб'єкта господарювання і зберігаємо зміни в бізнес-об'єкті до джерела даних.

Рівень служби містить інтерфейси з типовими операціями, такими як додавання, збереження, редагування та видалення. Крім того, цей шар використовується для зв'язку між шаром інтерфейсу та шаром сховища. Рівень обслуговування також може містити бізнес-логіку для сутності. У цьому шарі інтерфейси обслуговування зберігаються окремо від його реалізації, зберігаючи вільний зв'язок і поділ проблем.

Шар UI — найбільш зовнішній шар і зберігає периферійні проблеми, такі як інтерфейс та тести. Для веб-застосунку він представляє проект веб-API або проект тесту. Цей шар має реалізацію принципу інжекції залежностей так, що додаток будує

вільно зв'язану структуру і може передаватись до внутрішнього рівня через інтерфейси.

Керівництво з onion-архітектури не надає жодних напрямків щодо того, як повинні бути реалізовані шари. Архітектор повинен вирішити реалізацію і вільно обирати будь-який рівень класу, пакета, модуля або будь-чого іншого, що потрібно для додавання до рішення.

Нижче наведено переваги реалізації onion-архітектури:

— Пластини цибулі архітектури з'єднані через інтерфейси.

— Інжекція забезпечуються під час роботи.

— Архітектура додатків побудована поверх моделі домену.

— Вся зовнішня залежність, наприклад, доступ до бази даних і виклики сервісу, представлена у зовнішніх шарах.

— Відсутність залежностей внутрішнього шару від зовнішніх шарів.

— Залежності в напрямку центру.

— Гнучка і стійка портативна архітектура.

— Немає необхідності створювати спільні та спільні проекти.

— Можна швидко перевірити, оскільки ядро програми не залежить ні від чого.

Кілька недоліків onion-архітектури:

— Нелегко зрозуміти для початківців.

— Архітектори в основному зіпсують розподіл обов'язків між шарами.

— Сильно використовуються інтерфейси.

Цибульна архітектура широко прийнята в промисловості. Це дуже потужний і тісно пов'язаний з двома іншими архітектурними стилями — Layered і Hexagonal. Цибульна архітектура більш приваблива для C# програмістів, ніж програмісти Java.

4.1.1 Концептуальна модель

Концептуальна модель[14] — це систематизований змістовний опис модельованої системи (або проблемної ситуації) неформальною мовою.

Неформалізований опис розроблюваної імітаційної моделі включає визначення основних елементів системи, що моделюється, їх характеристики і взаємодія між

елементами власною мовою. При цьому можуть використовуватися таблиці, графіки, діаграми і т.д. Неформалізований опис моделі необхідний як самим розробникам (при перевірці адекватності моделі, її модифікації і т.д.), так і для взаєморозуміння з фахівцями інших профілів.

Концептуальна модель містить вихідну інформацію для системного аналітика, що виконує формалізацію системи і використовує для цього певну методологію і технологію, тобто на основі неформалізованого опису здійснюється розробка більш суворого і докладного формалізованого опису.

Потім формалізований опис перетворюється в програму - імітатор відповідно до деякої методики чи технології програмування.

Концептуальна модель створеного програмного продукту відображена на рисунку 4.1.

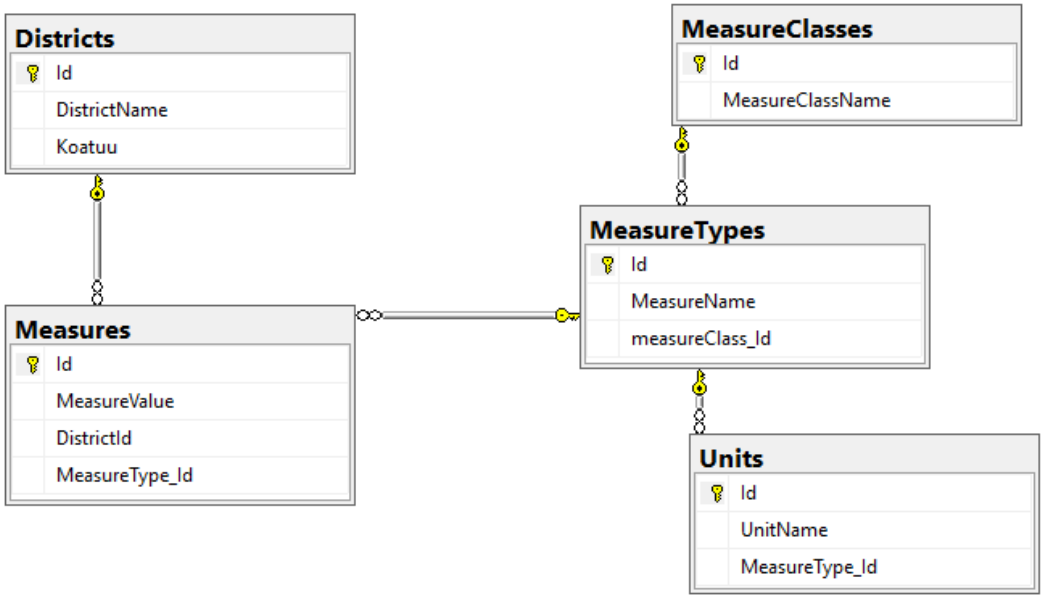


Рисунок 4.1 — Концептуальна модель

4.1.2 Шаблон MVP

Шаблон MVP[15] подібний шаблону MVC. Він виводиться з шаблону MVC, при цьому контролер замінюється представником.

Ця модель розділяє програму на три основні аспекти: модель, вид і представник.

Модель являє собою набір класів, що описує бізнес-логіку та дані. Вона також визначає бізнес-правила для даних, як дані можуть бути змінені і зманіпульовані.

Вид - це компонент, який безпосередньо взаємодіє з користувачем, подібним XML, Activity, fragments. Він не містить ніякої реалізованої логіки.

Представник отримує вхідні дані від користувачів за допомогою пункту "Вид", потім обробляє дані користувача за допомогою моделі та передає результати назад до перегляду. Представник спілкується з інтерфейсом перегляду. Інтерфейс визначається у класі представник, якому передаються необхідні дані. Активність, фрагмент або будь-який інший компонент перегляду реалізують цей інтерфейс і передають дані у бажаний спосіб (рисунок 4.2).

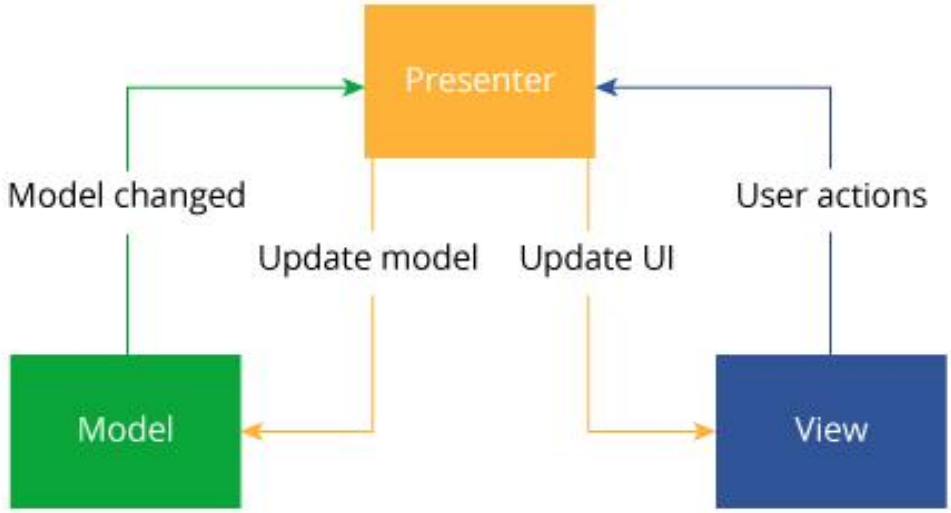


Рисунок 4.2 — Шаблон MVP

У шаблоні дизайну MVP представник маніпулює моделлю, а також оновлює подання. У MVP View і Presenter повністю відокремлені один від одного і спілкуються один з одним за допомогою інтерфейсу.

4.1.3 Шаблон Репозиторій

Шаблон Repository[16] набув досить великої популярності, оскільки він вперше був введений як частина Domain-Driven Design в 2004 році. Додавання, видалення,

оновлення та вибір елементів з цієї колекції здійснюється за допомогою низки простих методів, без необхідності мати справу з базами даних, і такими елементами, як зв’язки, команди, курсори або читачі.

Використання цього шаблону може допомогти у досягненні вільної взаємодії і може зберігати персистентність предметів домену необізнаними. Хоча картина дуже популярна (або, можливо, через це), вона також часто неправильно розуміється і неправильно використовується. Існує багато різних способів реалізації шаблону сховища.

Найпростіший підхід, особливо з існуючою системою, полягає у створенні нової реалізації сховища для кожного бізнес-об'єкту, який потрібно зберегти або отримати з вашого шару доступу до даних. Крім того, необхідно реалізовувати лише конкретні методи, які використовуються. Треба уникати створення "стандартного" класу сховища, базового класу або інтерфейсу за промовчанням, який необхідно реалізувати для всіх сховищ. Так, якщо потрібен метод Update або Delete, треба намагатися, щоб його інтерфейс був послідовним.

Інший підхід — створити простий, загальний інтерфейс для сховища. Також можна обмежити, з якими типами він працює, щоб нащадок був певного типу або реалізувати певний інтерфейс.

Перевага цього підходу полягає в тому, що він забезпечує спільний інтерфейс для роботи з будь-яким з об'єктів. Також можна спростити реалізацію за допомогою загальної реалізації сховища[17].

4.1.4 Шар доступу до даних

Шар доступу до даних (Data Access Layer - DAL) в програмному забезпеченні - це шар комп'ютерної програми, який надає спрощений доступ до даних, що зберігаються в постійному сховищі будь-якого типу, такому як реляційна база даних.

Реалізація шару доступу до даних у проекті виглядає наступним чином (рисунок 4.3).

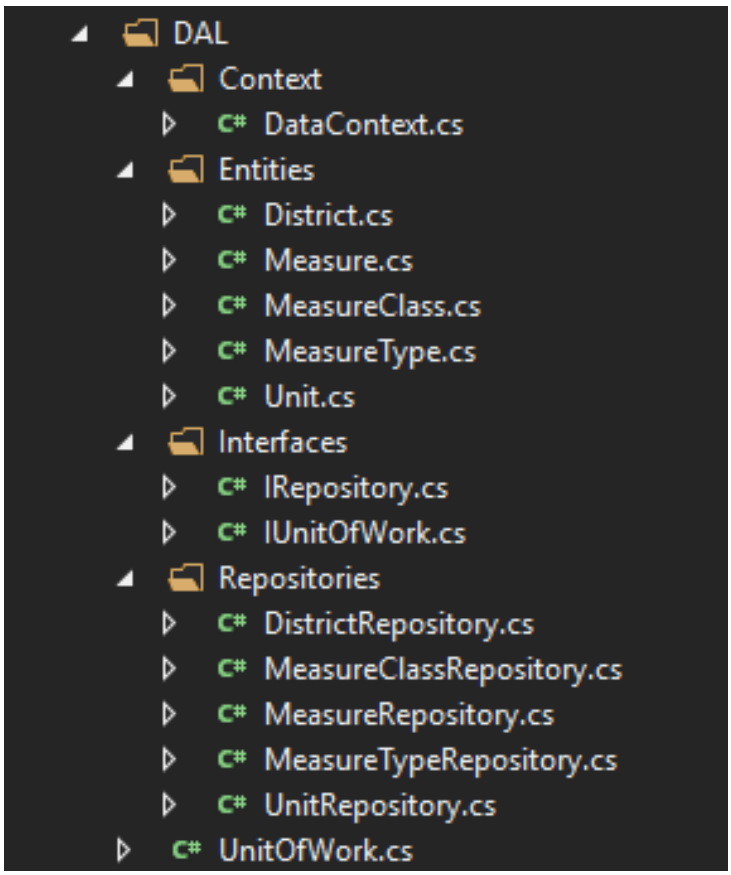


Рисунок 4.3 — Доступ до даних

Шар доступу до даних зберігає моделі, що описують використовувані суті. Тут також зберігаються репозиторії, через які рівень бізнес-логіки взаємодіє з базою даних.

4.1.5 Шар бізнес-логіки

Шар бізнес-логіки містить набір компонентів, які відповідають за обробку отриманих від рівня представлень даних, реалізує всю необхідну логіку додатка і взаємодію з базою даних (рисунок 4.4).

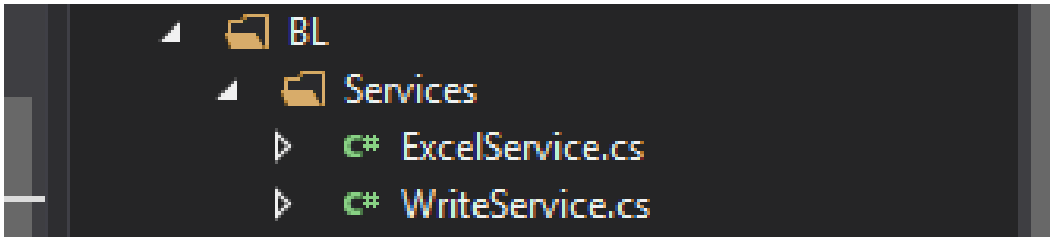


Рисунок 4.4 — Шар бізнес-логіки

На рівні шару бізнес-логіки реалізовано сервіс «ExcelService» який відповідає безпосередньо за роботу за файлами які обрав користувач.

«ExcelService» виконує відкриття файлу, конвертацію даних до проміжних типів та запису до програмного об’єкту, який і повертає в результаті своєї роботи.

Сервіс «WriteService» виконує запис до бази даних інформації яку відкрив та перевірив користувач.

4.1.4 Шар інтерфейсу користувача

Рівень представлення - рівень, з яким безпосередньо взаємодіє користувач. Цей рівень включає компоненти для користувача інтерфейсу, механізм отримання введення від користувача.

Представлено головне вікно програмного продукту (рисунок 4.5).

Рисунок 4.5 — Доступ до даних

4.2 Висновки до розділу

Для вирішення поставленої задачі було розроблено програмний продукт для моделювання оцінки соціально-політичних ризиків з використанням гіс технологій.

Також була представлена загальна структура програмного продукту, яка складається з декількох модулів, такі як:

— модуль для завантаження статистичних даних;

— модуль серверної частини та бази даних;

— модуль виведення статистики по обраній області;

— модуль відображення даних на мапі України.

Досягнуто збільшення зручності завдяки мінімізації і реалізації тільки необхідних інструментів, такий підхід полегшує розуміння програмного продукту з боку користувача та надає змогу працювати тільки з тим функціоналом, який потребує користувач.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1 Системні вимоги та інсталяція

Для запуску програмного продукту необхідна наявність заздалегідь встановлених пакетів платформи .NET Framework, версії не нижче 4.0, та

програмного комплексу ArcGis разом з деякими додатковими модулями такі, як Arc Object SDK[18] та ArcMap.

Програмний продукт не потребує встановлення на локальну машину, може бути запущений з використанням будь-якого носія.

Для нормальної роботи системи необхідно задоволення наступних вимог до апаратного забезпечення:

— процесор Intel Core i3 2.4 GHz;

— оперативна пам’ять в об’ємі 8 Гб:

— відео пам'ять R7 360 2 Гб.

5.2 Сценарії роботи з програмним продуктом

Запуск програмного додатку виконується запуском виконуючого файлу, який знаходиться в системному каталозі у якому зберігаються усі модулі програмного комплексу. При запуску відображається головне вікно програми (рисунок 5.1). На якому на першому етапі не буде відображена жодна інформація.

Для початку роботи, користувачу необхідно натиснути кнопку «Open xls», та після цього обрати необхідний для завантаження статистичних даних файл з розширенням *.xlsx (рисунок 5.2).

Рисунок 5.1 — Головне вікно

Відкриття файлу виконує користувач при натисканні на кнопку «Open File»

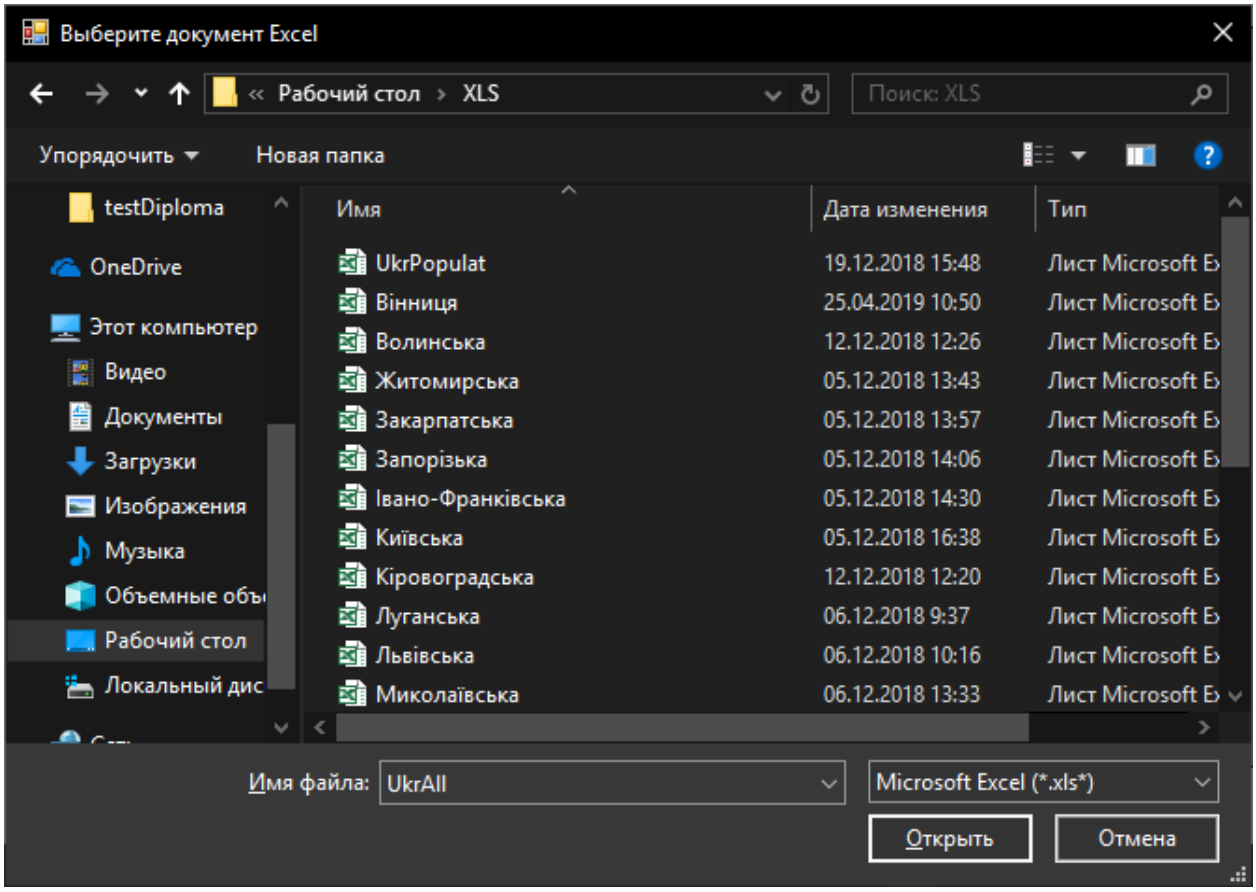
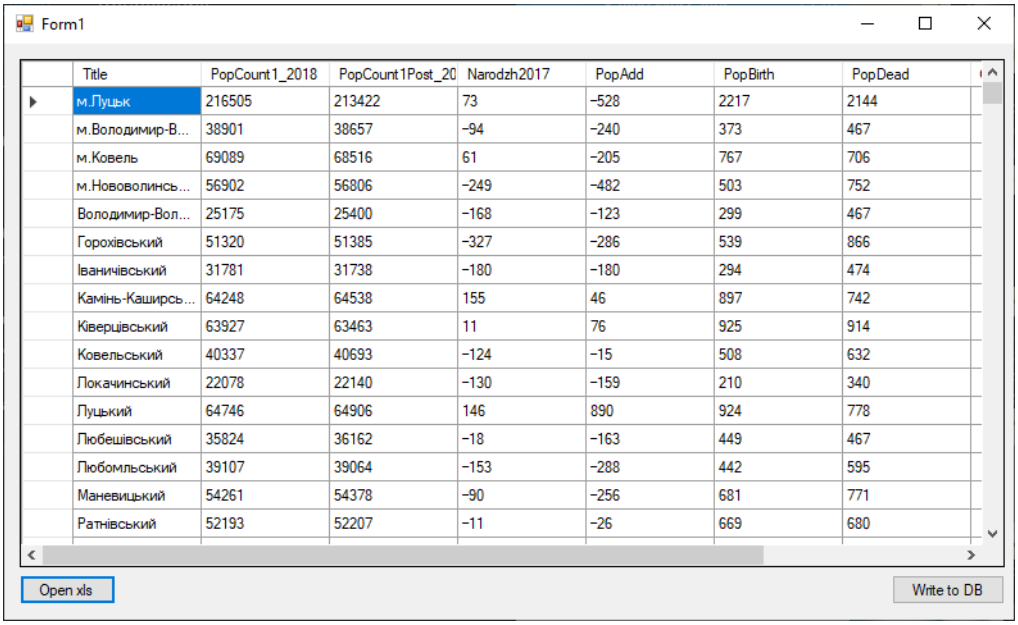


Рисунок 5.2 — Відкриття файлу

Після того, як користувач обрав потрібний файл, та підтвердив свою дію – файл завантажуєтеся і відображається у вікні програми (рисунок 5.3).



Title	PopCount1_2018	PopCount1Post_20	Narodzh2017	PopAdd	PopBirth	PopDead
м.Луцьк	216505	213422	73	-528	2217	2144
м.Володимир-В...	38901	38657	-94	-240	373	467
м.Ковель	69089	68516	61	-205	767	706
м.Нововолинсь...	56902	56806	-249	-482	503	752
Володимир-Вол...	25175	25400	-168	-123	299	467
Горохівський	51320	51385	-327	-286	539	866
Іваничівський	31781	31738	-180	-180	294	474
Камінь-Каширсь...	64248	64538	155	46	897	742
Ківерцівський	63927	63463	11	76	925	914
Ковельський	40337	40693	-124	-15	508	632
Локачинський	22078	22140	-130	-159	210	340
Луцький	64746	64906	146	890	924	778
Любешівський	35824	36162	-18	-163	449	467
Любомльський	39107	39064	-153	-288	442	595
Маневицький	54261	54378	-90	-256	681	771
Ратнівський	52193	52207	-11	-26	669	680

Рисунок 5.3 — Результат завантаження даних

Після того, як користувач впевнився в коректності завантажених даних, потрібно натиснути програмну кнопку «Write to DB», завдяки якій усі дані будуть розбиті на окремі структури і записані до відповідних комірок в базі даних. Після відпрацювання, користувачу буде виведене відповідне вікно з інформацію про успішний запис (рисунок 5.4).

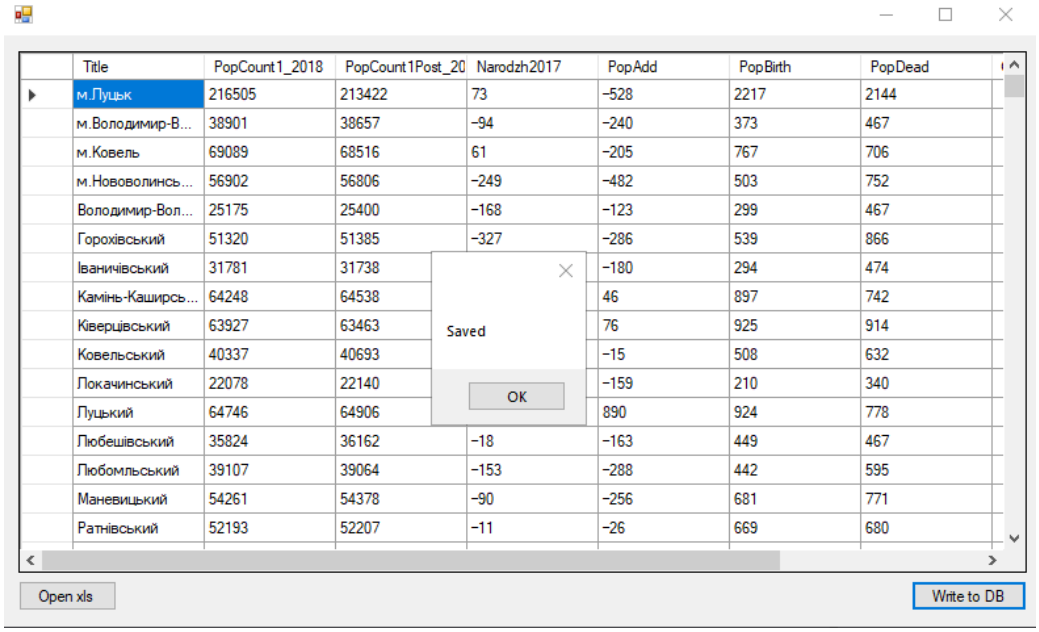


Рисунок 5.4 — Результат збереження даних

Для відображення отриманих даних необхідно запустити іншу частину програми. Для запуску необхідно відкрити виконуючий файл Listdistrict.exe.

Після завантаження програми буде відображено декілька вікон, на одній з яких відображається мапа України, а на іншій – статистичні дані по обраному регіону.

Коли користувач обирає певну область, на мапі ця область відображається іншим кольором, а на формі зі статистикою виводяться дані про обрану область та район (рисунок 5.5).

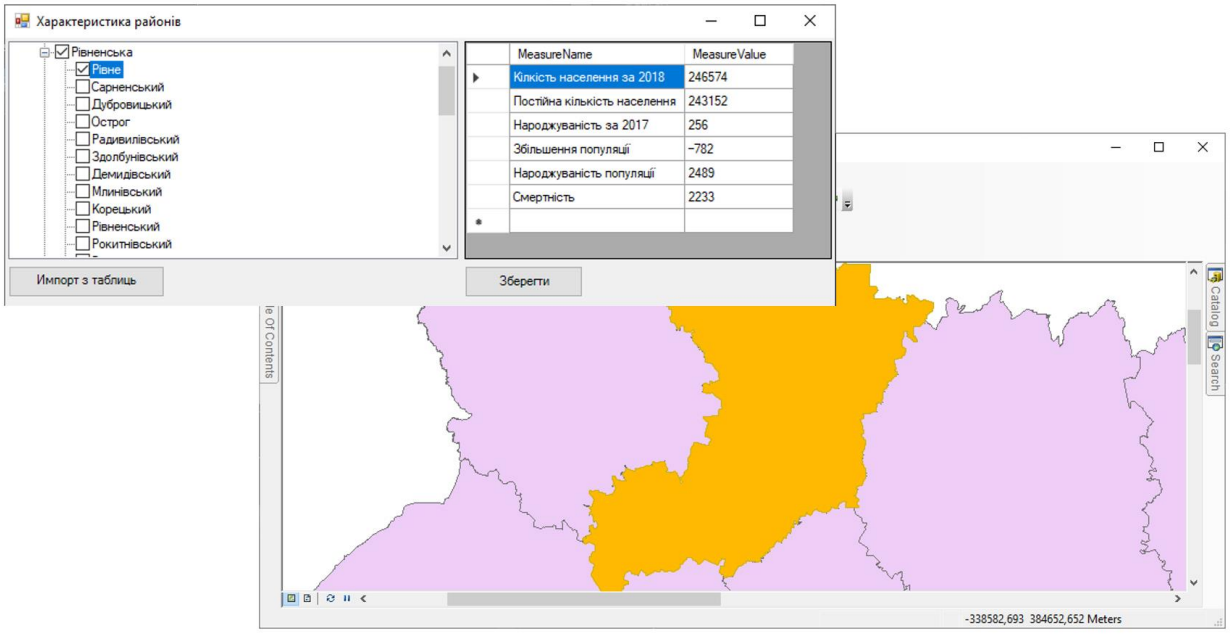


Рисунок 5.5 — Результат відображення даних

ВИСНОВКИ

Результатом виконання дипломного проекту стала розробка програмного застосунку для завантаження та відображення статистичних даних щодо демографічної безпеки України. Програмний продукт розроблений об’єктно-

орієнтованою мовою програмування C#, програмної платформи .NET framework. В якості середовища розробки використано програмне забезпечення Visual Studio 2017.

Було проведено дослідження існуючих програмних комплексів для побудови геоінформаційних систем, проаналізовані основні недоліки, запропоновані програмні та інтерфейсні рішення для покращення роботи.

Розроблене програмне забезпечення надає користувачу можливість завантажити файл зі статистичними даними з комп’ютера, записати їх до бази даних та відобразити їх на мапі України.

Необхідними можливостями, які забезпечує додаток, є:

— завантаження даних з комп’ютера користувача;

— відкриття та зчитування файлів у форматі *.xlsx;

— можливість перегляду завантажених даних;

— декомпозиція та запис даних до бази даних;

— можливість відобразити дані на мапі України.

В ході роботи було поглиблено знання та навички використання об'єктно-орієнтованої мови, створення додатку від ідеї до повноцінного продукту дала можливість пройти всі етапи розробки і опрацювати різні аспекти: БД, візуальна та функціональна складові і тд.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Плоский дизайн»: с чего начать? Пять основных принципов Flat дизайна [Електронний ресурс] – Режим доступу до ресурсу: http://powerbranding.ru/design/flat-design-june13/.

2. Цвігун І.А Демографічна безпека України та напрям її регулювання / Цвігун І.А – 2013.
3. «Альтернативы для замены ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://ruprogi.ru/software/arcgisdesktop.
4. «.NET Framework Guide» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/
5. «Common Language Runtime (CLR) overview» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/standard/clr
6. Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – Киев: Питер, 2008. – 656 с. – (2).
7. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: вильямс, 2014. – 1312 с. – (6).
8. «Visual Studio» [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/
9. «Windows Forms overview» [Електронний ресурс] –
10. Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/framework/winforms/windows-forms-overview
11. «Entity Framework Documentation» [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/
12. «SQL Server 2017» [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/ru-ru/sql-server/sql-server-2017
13. «About ArcGIS» [Електронний ресурс] – Режим доступу до ресурсу: https://www.esri.com/ru-ru/arcgis/about-arcgis/overview
14. «Understanding Onion Architecture» [Електронний ресурс] – Режим доступу до ресурсу: https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understanding-onion-architecture.html
15. «Концептуальная модель системы» [Електронний ресурс] – Режим доступу до ресурсу: http://studepedia.org/index.php?vol=1&post=2123

16. «MVC, MVP and MVVM Design Pattern» [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad
17. «UnitOfWork And Repository Pattern » [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern-305cd8ecfa7a
18. «What is ArcGIS, and where is it available at IU?» [Електронний ресурс] – Режим доступу до ресурсу: https://kb.iu.edu/d/avzt

ДОДАТОК 1

Моделювання оцінки соціально-політичних ризиків з використанням ГІС технологій

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 81-1	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-1	Diploma.cs	Модулі клієнтської частини програмного продукту
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 12-2	Window.cs	Модуль інтерфейсу клієнтської частини
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТМ52 19Б 13-1	Опис.docx	Опис модуля інтерфейсу клієнтської частини програми

ДОДАТОК 2

Моделювання оцінки соціально-політичних ризиків з використанням ГІС технологій

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52

Аркушів 10

Київ – 2019

```
using System;
```

```
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;

namespace Diploma.BL.Services
{
   class ExcelService
   {

      public DataTable GetDataFromExcel(string fileName)
      {
         //рабоата с Excel
         Excel.Range range;
         Excel.Workbook xlWB;
         Excel.Worksheet xlSht;
         int iLastRow, iLastCol;

         Excel.Application xlApp = new Excel.Application(); //создаём приложение Excel
         xlWB = xlApp.Workbooks.Open(fileName); //открываем наш файл
         xlSht = xlWB.Worksheets["Лист1"]; //или так xlSht = xlWB.ActiveSheet
//активный лист

         iLastRow = xlSht.Cells[xlSht.Rows.Count,
"A"].End[Excel.XlDirection.xlUp].Row; //последняя заполненная строка в столбце A
         iLastCol = xlSht.Cells[1,
xlSht.Columns.Count].End[Excel.XlDirection.xlToLeft].Column; //последний
заполненный столбец в 1-й строке

         range = (Excel.Range)xlSht.Range["A1", xlSht.Cells[iLastRow, iLastCol]];
//пример записи диапазона ячеек в переменную Rng
                                                //Rng = xlSht.get_Range("A1",
"B10"); //пример записи диапазона ячеек в переменную Rng
                                                //Rng = xlSht.get_Range("A1:B10");
//пример записи диапазона ячеек в переменную Rng
                                                //Rng = xlSht.UsedRange; //пример
записи диапазона ячеек в переменную Rng

         var dataArr = (object[,])range.Value; //чтение данных из ячеек в массив

//xlSht.get_Range("K1").get_Resize(dataArr.GetUpperBound(0),
dataArr.GetUpperBound(1)).Value = dataArr; //выгрузка массива на лист
```

```
    //закрытие Excel
    xlWB.Close(true); //сохраняем и закрываем файл
    xlApp.Quit();
    releaseObject(xlSht);
    releaseObject(xlWB);
    releaseObject(xlApp);

    //заполняем DataTable для последующего заполнения dataGridView
    DataTable dt = new DataTable();
    DataRow dtRow;

    //добавляем столбцы в DataTable
    for (int i = 1; i <= dataArr.GetUpperBound(1); i++)
        //dt.Columns.Add((string)dataArr[1, i]);
        dt.Columns.Add(Convert.ToString(dataArr[1, i]));

    //цикл по строкам массива
    for (int i = 2; i <= dataArr.GetUpperBound(0); i++)
    {
        dtRow = dt.NewRow();
        //цикл по столбцам массива
        for (int n = 1; n <= dataArr.GetUpperBound(1); n++)
        {
            dtRow[n - 1] = dataArr[i, n];
        }
        dt.Rows.Add(dtRow);
    }

    return dt;
}

private void releaseObject(object obj)
{
    try
    {
        System.Runtime.InteropServices.Marshal.ReleaseComObject(obj);
        obj = null;
    }
    catch (Exception ex)
    {
        obj = null;
        MessageBox.Show("Unable to release the Object " + ex.ToString());
    }
    finally
```

```
            {
                GC.Collect();
            }
        }
    }
}
using Diploma.DAL;
using Diploma.Repositories;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Diploma.BL.Services
{
    class WriteService
    {
        private DistrictRepository districtRepository;
        private MeasureRepository measureRepository;
        private MeasureClassRepository measureClassRepository;
        private MeasureTypeRepository measureTypeRepository;
        private UnitRepository unitRepository;

        private UnitOfWork unitOfWork;

        public WriteService(UnitOfWork unitOfWork)
        {
            this.unitOfWork = unitOfWork;

            //districtRepository = (DistrictRepository)unitOfWork.Districts;
            //measureRepository = (MeasureRepository)unitOfWork.Measures;
            //measureClassRepository =
(MeasureClassRepository)unitOfWork.MeasureClasses;
            //measureTypeRepository = (MeasureTypeRepository)unitOfWork.MeasureTypes;
            //unitRepository = (UnitRepository)unitOfWork.Units;

            districtRepository = unitOfWork.Districts as DistrictRepository;
            measureRepository = unitOfWork.Measures as MeasureRepository;
            measureClassRepository = unitOfWork.MeasureClasses as
MeasureClassRepository;
            measureTypeRepository = unitOfWork.MeasureTypes as MeasureTypeRepository;
            unitRepository = unitOfWork.Units as UnitRepository;
        }
```

```
        public void WriteToDb(DataTable dataTable)
        {
            SetMeasureType();

            DataRow dataRow;
            District district;



            for (int i = 0; i < dataTable.Rows.Count; i++)
            {
                dataRow = dataTable.Rows[i];

                district = new District
                {
                    DistrictName = dataRow[0].ToString(),
                    Koatuu = dataRow[8].ToString()
                };

                districtRepository.Create(district);
                unitOfWork.Save();

                measureRepository.Create(new Measure
                {
                    MeasureValue = dataRow[1].ToString(),
                    MeasureType = measureTypeRepository.Find(x => x.MeasureName ==
"Кількість населення за 2018").First(),
                    District = district

                });
                measureRepository.Create(new Measure
                {
                    MeasureValue = dataRow[2].ToString(),
                    MeasureType = measureTypeRepository.Find(x => x.MeasureName ==
"Постійна кількість населення").First(),
                    District = district
                });
                measureRepository.Create(new Measure
                {
                    MeasureValue = dataRow[3].ToString(),
                    MeasureType = measureTypeRepository.Find(x => x.MeasureName ==
"Народжуваність за 2017").First(),
                    District = district
                });
                measureRepository.Create(new Measure
```

```
            {

                MeasureValue = dataRow[4].ToString(),
                MeasureType = measureTypeRepository.Find(x => x.MeasureName ==
"Збільшення популяції").First(),
                District = district
            });
            measureRepository.Create(new Measure
            {
                MeasureValue = dataRow[5].ToString(),
                MeasureType = measureTypeRepository.Find(x => x.MeasureName ==
"Народжуваність популяції").First(),
                District = district
            });
            measureRepository.Create(new Measure
            {
                MeasureValue = dataRow[6].ToString(),
                MeasureType = measureTypeRepository.Find(x => x.MeasureName ==
"Смертність").First(),
                District = district
            });

            unitOfWork.Save();
        }
        unitOfWork.Dispose();
    }

    private void SetMeasureType()
    {
        SetUnit();
        SetMeasureClass();
if (!measureTypeRepository.GetAll().Any())
        {
            measureTypeRepository.Create(new MeasureType {
                measureClass = measureClassRepository.Find(x => x.MeasureClassName ==
"Население").First(),
                Units = unitRepository.Find(x => x.UnitName == "Человек").ToList(),
                MeasureName = "Кількість населення за 2018"
            });

            measureTypeRepository.Create(new MeasureType
            {
                measureClass = measureClassRepository.Find(x => x.MeasureClassName ==
"Население").First(),
                Units = unitRepository.Find(x => x.UnitName == "Человек").ToList(),
```

```
            MeasureName = "Постійна кількість населення"
        });

        measureTypeRepository.Create(new MeasureType
        {
            measureClass = measureClassRepository.Find(x => x.MeasureClassName ==
"Население").First(),
            Units = unitRepository.Find(x => x.UnitName == "Человек").ToList(),
            MeasureName = "Народжуваність за 2017"
        });

        measureTypeRepository.Create(new MeasureType
        {
            measureClass = measureClassRepository.Find(x => x.MeasureClassName ==
"Население").First(),
            Units = unitRepository.Find(x => x.UnitName == "Человек").ToList(),
            MeasureName = "Збільшення популяції"
        });

        measureTypeRepository.Create(new MeasureType
        {
            measureClass = measureClassRepository.Find(x => x.MeasureClassName ==
"Население").First(),
            Units = unitRepository.Find(x => x.UnitName == "Человек").ToList(),
            MeasureName = "Народжуваність популяції"
        });

        measureTypeRepository.Create(new MeasureType
        {
            measureClass = measureClassRepository.Find(x => x.MeasureClassName ==
"Население").First(),
            Units = unitRepository.Find(x => x.UnitName == "Человек").ToList(),
            MeasureName = "Смертність"
        });

        unitOfWork.Save();

    }
}
private void SetUnit()
{
    if (!unitRepository.GetAll().Any())
    {
        unitRepository.Create(new Unit
        {
```

```
                UnitName = "Человек"
            });

            unitOfWork.Save();
        }
    }
    private void SetMeasureClass()
    {
        if (!measureClassRepository.GetAll().Any())
        {
            measureClassRepository.Create(new MeasureClass
            {
                MeasureClassName = "Население"
            });

            unitOfWork.Save();
        }
    }

  }
}
using Diploma.Context;
using Diploma.Interfaces;
using Diploma.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Diploma.DAL
{
  public class UnitOfWork : IUnitOfWork
  {
    private DataContext context = new DataContext("DBConnection"); // NEED CON
STRING

    private DistrictRepository districtRepository;
    private MeasureRepository measureRepository;
    private MeasureClassRepository measureClassRepository;
    private MeasureTypeRepository measureTypeRepository;
    private UnitRepository unitRepository;

    public IRepository<District> Districts
    {
```

```
    get
    {
        if (districtRepository == null)
        {
            districtRepository = new DistrictRepository(context);
        }
        return districtRepository;
    }
}

public IRepository<Measure> Measures
{
    get
    {
        if (measureRepository == null)
        {
            measureRepository = new MeasureRepository(context);
        }
        return measureRepository;
    }
}

public IRepository<MeasureClass> MeasureClasses
{
    get
    {
        if (measureClassRepository == null)
        {
            measureClassRepository = new MeasureClassRepository(context);
        }
        return measureClassRepository;
    }
}

public IRepository<MeasureType> MeasureTypes
{
    get
    {
        if (measureTypeRepository == null)
        {
            measureTypeRepository = new MeasureTypeRepository(context);
        }
        return measureTypeRepository;
    }
}
```

```
        public IRepository<Unit> Units
        {
            get
            {
                if (unitRepository == null)
                {
                    unitRepository = new UnitRepository(context);
                }
                return unitRepository;
            }
        }

        private bool disposed = false;

        public virtual void Dispose(bool disposing)
        {
            if (!this.disposed)
            {
                if (disposing)
                {
                    context.Dispose();
                }
                this.disposed = true;
            }
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        public void Save()
        {
            context.SaveChanges();
        }

    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Diploma.Interfaces
{
    public interface IRepository<T> where T : class
    {
        IEnumerable<T> GetAll();
        T Get(int id);
        IEnumerable<T> Find(Func<T, Boolean> predicate);
        void Create(T item);
        void Update(T item);
        void Delete(int id);
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Diploma.Interfaces
{
    public interface IUnitOfWork : IDisposable
    {
        IRepository<District> Districts { get; }
        IRepository<Measure> Measures { get; }
        IRepository<MeasureClass> MeasureClasses { get; }
        IRepository<MeasureType> MeasureTypes { get; }
        IRepository<Unit> Units { get; }

        void Save();
    }
}
```

ДОДАТОК 3

Моделювання оцінки соціально-політичних ризиків з використанням ГІС технологій

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52 19Б 13-2

Аркушів 8

Київ – 2019

АНОТАЦІЯ

Розроблений додаток надає інформацію про демографічну безпеку України та статистичну інформацію про обрану область.

Програмне забезпечення дозволяє отримати статистичну інформацію у візуальному вигляді про область яку обирає користувач.

Клієнтський застосунок був розроблений в середовище Microsoft Visual Studio 2017 з використанням платформи .NET та об’єктно-орієнтованої мови програмування C#.

ЗМІСТ

1. Загальні відомості 4
2. Функціональне призначення 5
3. Опис стилістики 6
4. Використовувані технічні засоби 7
5. Вхідні і вихідні дані 8

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до теми дипломної роботи, програма має назву «Моделювання оцінки соціально-політичних ризиків з використанням ГІС технологій».

Програма працює локально на комп’ютері користувача та не потребує доступу до мережі інтернет, що забезпечує практичність та швидкість.

Застосунок був написаний мовою C#.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмний додаток створений за для вирішення проблеми обробки та відображення статистичних даних що до демографічної безпеки України.

Це було реалізовано за деяких функцій системи, а саме:

— завантаження даних з комп’ютера користувача;

— відкриття та зчитування файлів у форматі *.xlsx;

— можливість перегляду завантажених даних;

— декомпозиція та запис даних до бази даних;

— можливість відобразити дані на мапі України.

ОПИС СТИЛІСТИКИ

Головна ідея стилістики додатку — максимальна простота інтерфейсу. Разом з тим дизайн спроектовано в стилі Flat UI, що зараз є провідним стилем багатьох застосунків, в тому числі desktop-сервісів. Він базується на принципах швейцарського стилю, в базі якого лежить типографіка як спосіб виключити зайві деталі інтерфейсу. Простота та мінімалізм є основою візуального оформлення такого проекту та забезпечує легкість сприйняття інформації, а також інтуїтивно зрозумілий інтерфейс для користувачів.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для користування програмним додатком потрібно мати комп’ютер або ноутбук з операційною системою Windows.

Для повноцінного користування програмою потрібно, щоб на комп’ютері був встановлені пакети Microsoft .NET Framework 4 або версії вище.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є:

— *.xlsx файл зі статистичними даними;

Вихідними даними є:

— результати відображення даних.